Python 新员工教材

**楚广明2012**

**新浪微博：@疯狂制造**

**请关注我，以便在第一时间获取更新消息**

**借用了一些闫小勇\郑纳智\学无止境等等同志的文档，向同志们致敬！**

**因为时间久远，很多引用文档以找不到作者，如果您发现我引用了您的文档，请第一时间联系我**

**引用作者目录我放在以下网址：http://2.myrestful.sinaapp.com/**

目录

[一、 Python概述 9](#_Toc372708744)

[1) 第一个Python程序 10](#_Toc372708745)

[1. 为什么要学习Python 11](#_Toc372708746)

[2. 代码块与缩进的差异 11](#_Toc372708747)

[3. 语句结尾的差异 12](#_Toc372708748)

[4. 注释方法 12](#_Toc372708749)

[5. 入口方法 12](#_Toc372708750)

[6. import与using语句 12](#_Toc372708751)

[2) 小结 12](#_Toc372708752)

[二、 基本数据类型 13](#_Toc372708753)

[1) 第一个程序的解析 13](#_Toc372708754)

[2) 变量的命名规范 15](#_Toc372708755)

[3) 空类型 15](#_Toc372708756)

[4) 布尔类型 15](#_Toc372708757)

[5) 数值类型 16](#_Toc372708758)

[6) 字符串类型 16](#_Toc372708759)

[7) 全局变量 17](#_Toc372708760)

[三、 基本数据类型-列表(list【】) 18](#_Toc372708761)

[1) 概述 18](#_Toc372708762)

[2) 常用方法 20](#_Toc372708763)

[四、 基本数据类型-字典(dict｛｝) 20](#_Toc372708764)

[1) 概述 20](#_Toc372708765)

[2) 常用方法 22](#_Toc372708766)

[五、 基本数据类型-元组(tuple（）) 23](#_Toc372708767)

[六、 基本数据类型-集合(set) 24](#_Toc372708768)

[七、 数据类型转换 25](#_Toc372708769)

[1) 概述 25](#_Toc372708770)

[2) 常用方法 25](#_Toc372708771)

[八、 运算符、表达式和流程控制 26](#_Toc372708772)

[1) 算术运算符与算术表达式 26](#_Toc372708773)

[2) 赋值运算符与赋值表达式 26](#_Toc372708774)

[3) 关系运算符与关系表达式 27](#_Toc372708775)

[九、 流程控制 27](#_Toc372708776)

[1) 条件语句 27](#_Toc372708777)

[2) 循环 28](#_Toc372708778)

[十、 异常 29](#_Toc372708779)

[十一、 动态表达式 30](#_Toc372708780)

[十二、 函数及函数编程 31](#_Toc372708781)

[1) 函数的定义 31](#_Toc372708782)

[2) 定义一个交换函数 31](#_Toc372708783)

[3) 函数的默认参数与返回值 32](#_Toc372708784)

[4) 返回多个值 32](#_Toc372708785)

[5) locals函数和globals函数介绍 32](#_Toc372708786)

[十三、 类及面向对象 33](#_Toc372708787)

[1) 类的定义 33](#_Toc372708788)

[2) 为类添加数据 34](#_Toc372708789)

[3) 构造函数 34](#_Toc372708790)

[4) 类的访问修饰符 35](#_Toc372708791)

[5) 为类添加方法 36](#_Toc372708792)

[6) 静态方法 37](#_Toc372708793)

[7) 单继承 37](#_Toc372708794)

[8) 传值与传引用 38](#_Toc372708795)

[十四、 模块与包 39](#_Toc372708796)

[1) 对于模块的理解 40](#_Toc372708797)

[2) 模块的显要特性：属性 40](#_Toc372708798)

[3) 模块创建过程的例子 41](#_Toc372708799)

[4) 模块的导入 41](#_Toc372708800)

[5) 模块的属性 42](#_Toc372708801)

[十五、 字符串与简单的正则表达式 42](#_Toc372708802)

[1) 解决中文字符的问题 43](#_Toc372708803)

[2) 单双引号的区别 44](#_Toc372708804)

[3) 字符串的格式化 44](#_Toc372708805)

[4) 字符串的合并与分割 46](#_Toc372708806)

[5) 常用字符操作演示 47](#_Toc372708807)

[十六、 文件IO 50](#_Toc372708808)

[1) 基本文件功能演示 50](#_Toc372708809)

[2) 文件的写入 52](#_Toc372708810)

[3) 文件的删除与复制 52](#_Toc372708811)

[4) 文件与目录的重命名 53](#_Toc372708812)

[5) 文件内容的查找和替换 54](#_Toc372708813)

[6) 文件的比较 55](#_Toc372708814)

[7) 控制台输入 55](#_Toc372708815)

[十七、 中文乱码 55](#_Toc372708816)

[1) 编码普及 56](#_Toc372708817)

[1. ASCII码 56](#_Toc372708818)

[2. 非ASCII编码 56](#_Toc372708819)

[3. Unicode 56](#_Toc372708820)

[4. Unicode的问题 56](#_Toc372708821)

[5. UTF-8 57](#_Toc372708822)

[6. Unicode与UTF-8之间的转换 58](#_Toc372708823)

[7. Little endian和Big endian 58](#_Toc372708824)

[2) 中文乱码细究 59](#_Toc372708825)

[1. python中的str和unicode 59](#_Toc372708826)

[2. str和unicode对象的转换 60](#_Toc372708827)

[3) Setdefaultencoding 60](#_Toc372708828)

[3. 操作不同文件的编码格式的文件 60](#_Toc372708829)

[4. 文件的编码格式和编码声明的作用 61](#_Toc372708830)

[4) 新手编码必读（gonewithewind大虾作品） 62](#_Toc372708831)

[十八、 Web.Py框架概述 63](#_Toc372708832)

[1) Python下web开发框架的选择 63](#_Toc372708833)

[1. Django 63](#_Toc372708834)

[2. Pylons & TurboGears & repoze.bfg 64](#_Toc372708835)

[3. Tornado & web.py 65](#_Toc372708836)

[4. Bottle & Flask 66](#_Toc372708837)

[5. Quixote 66](#_Toc372708838)

[6. 最后关于框架选择的误区 66](#_Toc372708839)

[2) 解决SAE第三方包的安装方案 67](#_Toc372708840)

[3) web.py安装 69](#_Toc372708841)

[4) web.py下常用框架简介 69](#_Toc372708842)

[5) 最简单的HelloWorld! 69](#_Toc372708843)

[6) Web.py基本使用模板使用 70](#_Toc372708844)

[7) 进阶模板使用 71](#_Toc372708845)

[1. server2.py 71](#_Toc372708846)

[2. post.html 71](#_Toc372708847)

[8) URL控制 72](#_Toc372708848)

[1. 问题: 72](#_Toc372708849)

[2. 解决: 72](#_Toc372708850)

[3. 路径匹配 72](#_Toc372708851)

[9) 一个完整的小例子 73](#_Toc372708852)

[10) web.seeother 和 web.redirect转向 73](#_Toc372708853)

[1. 问题 73](#_Toc372708854)

[2. 解法 73](#_Toc372708855)

[3. 区别 73](#_Toc372708856)

[11) 包含应用 74](#_Toc372708857)

[1. 实现 74](#_Toc372708858)

[12) 使用XML 75](#_Toc372708859)

[1. 问题 75](#_Toc372708860)

[2. 解法 75](#_Toc372708861)

[13) 获取POST数据 75](#_Toc372708862)

[1. login.html模板 76](#_Toc372708863)

[2. 主程序 76](#_Toc372708864)

[14) 获取客户端信息 77](#_Toc372708865)

[1. 问题 77](#_Toc372708866)

[2. 解法 77](#_Toc372708867)

[3. 例子 77](#_Toc372708868)

[4. 'ctx'中的数据成员 77](#_Toc372708869)

[15) 文件上传 78](#_Toc372708870)

[1. 简单的文件上传 78](#_Toc372708871)

[2. 基于新浪云的上传 79](#_Toc372708872)

[十九、 Web.py使用Session 80](#_Toc372708873)

[1) 问题 80](#_Toc372708874)

[2) 解法 80](#_Toc372708875)

[3) 在template下使用Session 82](#_Toc372708876)

[1. 问题: 82](#_Toc372708877)

[2. 解决: 82](#_Toc372708878)

[4) 如何操作Cookies 83](#_Toc372708879)

[1. 问题 83](#_Toc372708880)

[2. 解法 83](#_Toc372708881)

[3. 设置Cookies 83](#_Toc372708882)

[4. 获得Cookies 84](#_Toc372708883)

[5) 布局模板 85](#_Toc372708884)

[1. 问题 85](#_Toc372708885)

[2. 方法 85](#_Toc372708886)

[3. Tip: 在布局文件（layout.html）中定义的页面标题变量 86](#_Toc372708887)

[4. Tip: 在其他模板中引用css文件，如下: 87](#_Toc372708888)

[二十、 Web.py支持的语法模板 87](#_Toc372708889)

[1) 使用Templetor模板语法 87](#_Toc372708890)

[1. Introduction 87](#_Toc372708891)

[2. 使用模板系统 88](#_Toc372708892)

[3. 表达式用法 89](#_Toc372708893)

[4. 赋值 89](#_Toc372708894)

[5. 过滤 90](#_Toc372708895)

[6. 新起一行用法 90](#_Toc372708896)

[7. 转义 $ 90](#_Toc372708897)

[8. 注释 90](#_Toc372708898)

[9. 控制结构 90](#_Toc372708899)

[10. 使用 def 91](#_Toc372708900)

[11. 使用 var 92](#_Toc372708901)

[12. 内置 和 全局 93](#_Toc372708902)

[2) 使用Jinja2模板语法 94](#_Toc372708903)

[1. 上手的一个小例子 94](#_Toc372708904)

[二十一、 Web.py中使用数据库 95](#_Toc372708905)

[1) Mysql-Python在Ubuntu下的安装 95](#_Toc372708906)

[2) 使用web.py自带的DB封装库 95](#_Toc372708907)

[1. 一个简单的数据库访问测试 95](#_Toc372708908)

[2. 一个复杂一点的例子 96](#_Toc372708909)

[3. 更加复杂的例子 96](#_Toc372708910)

[二十二、 使用Sqlalchemy 98](#_Toc372708911)

[1) 简介 98](#_Toc372708912)

[2) 初始化Sqlite数据库 100](#_Toc372708913)

[3) 插入数据库 101](#_Toc372708914)

[4) 在视图中使用sqlalchemy 104](#_Toc372708915)

[1. 首先创建数据库models.py 104](#_Toc372708916)

[2. 使用app.py 105](#_Toc372708917)

[二十三、 Json的使用 106](#_Toc372708918)

[二十四、 Socket编程 107](#_Toc372708919)

[二十五、 Urllib模块 107](#_Toc372708920)

[1) urlopen的基本使用 107](#_Toc372708921)

[2) urlretrieve方法的使用 108](#_Toc372708922)

[3) url编码 109](#_Toc372708923)

[4) 使用httplib抓取 110](#_Toc372708924)

[二十六、 Urllib2的使用 112](#_Toc372708925)

[1) 最简单的爬虫 112](#_Toc372708926)

[2) 提交表单数据 112](#_Toc372708927)

[1. 用GET方法提交数据 112](#_Toc372708928)

[二十七、 Html的解析 112](#_Toc372708929)

[1) BeautifulSoup进行解析 113](#_Toc372708930)

[2) 使用PyQuery来进行XML与Html的解析 114](#_Toc372708931)

[1. 快速入门 114](#_Toc372708932)

[2. 元素操作 115](#_Toc372708933)

[3. 其他操作 115](#_Toc372708934)

[二十八、 操作Memcached 116](#_Toc372708935)

[1) 安装 117](#_Toc372708936)

[1. Linux 环境 117](#_Toc372708937)

[2. Win环境 117](#_Toc372708938)

[3. linux启动memcached 117](#_Toc372708939)

[4. windows下启动 118](#_Toc372708940)

[2) Python操作Memcached 118](#_Toc372708941)

[1. Python操作memcached 118](#_Toc372708942)

[2. Memcached常用方法 119](#_Toc372708943)

[二十九、 SQlite模块 119](#_Toc372708944)

[1) 简单的介绍 119](#_Toc372708945)

[2) 安装与使用 119](#_Toc372708946)

[3) 一个简单的例子 121](#_Toc372708947)

[4) 中文处理 124](#_Toc372708948)

[三十、 正则表达式 124](#_Toc372708949)

[1) 简介 124](#_Toc372708950)

[2) 3个重要的正则式命令 125](#_Toc372708951)

[1. 验证字符串 127](#_Toc372708952)

[2. 验证url 127](#_Toc372708953)

[3. 验证Email 128](#_Toc372708954)

[4. 验证值在0-25的数字 128](#_Toc372708955)

[5. 验证格式为MM/DD/YYYY, YYYY/MM/DD and DD/MM/YYYY的日期 128](#_Toc372708956)

[3) 正则表达符 129](#_Toc372708957)

[1. 你常用的元字符 129](#_Toc372708958)

[2. 常用的限定符 129](#_Toc372708959)

[3. 字符串类 130](#_Toc372708960)

[4. 分枝条件 131](#_Toc372708961)

[5. 分组 131](#_Toc372708962)

[6. 反义 132](#_Toc372708963)

[7. 后向引用 132](#_Toc372708964)

[8. 贪婪与懒惰 133](#_Toc372708965)

[4) 在Python中使用 134](#_Toc372708966)

[5) Re模块 136](#_Toc372708967)

[6) Match方法 138](#_Toc372708968)

[7) Pattern方法 139](#_Toc372708969)

[三十一、 Django框架 144](#_Toc372708970)

[1) 安装 144](#_Toc372708971)

[2) 入门 144](#_Toc372708972)

[三十二、 Python中的进程与线程 145](#_Toc372708973)

[1) 进程和线程 145](#_Toc372708974)

[2) 进程的运行环境 146](#_Toc372708975)

[3) 创建进程 146](#_Toc372708976)

[1. system函数 146](#_Toc372708977)

[2. exec家族函数 146](#_Toc372708978)

[4) 终止进程 147](#_Toc372708979)

[1. sys.exit函数 147](#_Toc372708980)

[2. os.abort函数 147](#_Toc372708981)

[5) 使用subprocess模块管理进程 147](#_Toc372708982)

[1. 使用Popen类管理进程 147](#_Toc372708983)

[6) 多线程概述 148](#_Toc372708984)

[1. 什么是多线程 148](#_Toc372708985)

[2. 线程的状态 148](#_Toc372708986)

[3. Python中的线程支持 149](#_Toc372708987)

[7) 生成和终止线程 149](#_Toc372708988)

[1. 使用thread模块 149](#_Toc372708989)

[2. thread的锁机制 150](#_Toc372708990)

[8) 使用threading.Thread类 151](#_Toc372708991)

[1. join等待线程完成 152](#_Toc372708992)

[2. 线程中的局部变量 153](#_Toc372708993)

[9) 线程之间的同步 153](#_Toc372708994)

[1. 临界资源和临界区 153](#_Toc372708995)

[三十三、 Pyexe 155](#_Toc372708996)

[三十四、 Sina SDK的使用 156](#_Toc372708997)

[1) 首先申请SDK开发者 156](#_Toc372708998)

[2) 下载Python SDK 158](#_Toc372708999)

[3) 一个简单的演示程序 158](#_Toc372709000)

[4) 一个进阶的演示 159](#_Toc372709001)

[三十五、 virtualenv 161](#_Toc372709002)

[1) 安装virtualenv 162](#_Toc372709003)

[2) 扩展virtualenv的工具 163](#_Toc372709004)

[3) python多版本管理利器-pythonbrew 164](#_Toc372709005)

[三十六、 附录-Ubuntu开发环境之安装 165](#_Toc372709006)

[1) 进入 live cd 桌面 165](#_Toc372709007)

[2) 安装系统 167](#_Toc372709008)

[3) 连网换源 176](#_Toc372709009)

[4) 更新系统 183](#_Toc372709010)

[三十七、 附录Ubuntu下的小工具 188](#_Toc372709011)

[1) 测试工具speedometer 188](#_Toc372709012)

[2) APT的使用 188](#_Toc372709013)

[三十八、 附录Ubuntu下的SVN创建 190](#_Toc372709014)

[1) 安装subversion 190](#_Toc372709015)

[2) 配置文件例子 192](#_Toc372709016)

[3) Hooks的使用 192](#_Toc372709017)

[三十九、 附录Sublimetext的使用 194](#_Toc372709018)

[1) 安装Sublime Text 2插件的方法： 194](#_Toc372709019)

[1. 直接安装 194](#_Toc372709020)

[2. 使用Package Control组件安装 194](#_Toc372709021)

[四十、 附录Windows下使用SVN Server 195](#_Toc372709022)

[1) VisualSVN Server的配置和使用方法【服务器端】 195](#_Toc372709023)

[2) TotoiseSVN的基本使用方法 199](#_Toc372709024)

[四十一、 附录vmware中的bridge、nat区别 201](#_Toc372709025)

[1) bridged(桥接模式) 201](#_Toc372709026)

[2) host-only(主机模式) 201](#_Toc372709027)

[3) NAT(网络地址转换模式) 202](#_Toc372709028)

[4) VMware网络连接的几种方式 202](#_Toc372709029)

[四十二、 附录 Python学习资源大全 206](#_Toc372709030)

[1) 学会使用搜索引擎，尤其是Google。推荐阅读文章： 206](#_Toc372709031)

[2) 学会提问 207](#_Toc372709032)

[3) 书籍推荐： 207](#_Toc372709033)

[4) 视频推荐： 207](#_Toc372709034)

[5) 学习方法 207](#_Toc372709035)

[1. *知乎问答*：零基础，应当如何开始学习 Python ？点击打开链接--by@黎敏 207](#_Toc372709036)

[2. *芝麻问答*：如何学习Python 点击打开链接作者：@halida 208](#_Toc372709037)

[3. *知乎问答*：你是如何自学 Python 的？点击打开链接 作者:@成增存 210](#_Toc372709038)

[6) Just Do It 210](#_Toc372709039)

[7) 更新 210](#_Toc372709040)

[四十三、 附录-常用Python模块 211](#_Toc372709041)

# Python概述

Python是一门优雅而健壮的编程语言，它继承了传统编译语言的强大性与通用性，同时也借鉴了简单的脚本和解释语言的易用性。它可以帮你完成工作，而且一段时间之后，你还能看明白自己写的这段代码。你会对自己如此快地学会它和它强大的功能感到十分的惊讶，更不用提你已经完成了工作了，只有你想不到，没有Python做不到的。

就算你的项目中有大量的Python代码，你也依旧可以有条不紊地通过将其分离为多个文件或模块加以组织管理。而且你可以从一个模块中选取代码，而从另一个模块中读取属性。更棒的是，对于所有模块，Python的访问语法都是相同的。不管这个模块是Python标准库中的还是你一分钟前创建的，哪怕是你用其他语言写的扩展都没有问题！借助这些特点，你会自已根据需要扩展了这门语言，而且你也这么做了。

代码中的瓶颈可能是在性能分析中总排在前面的那些热门或者一些特别强调性能的地方，可以作为Python扩展用C重写。需要重申的是，这些接口和纯Python模块的接口是一模一样的，乃至代码和对象的访问方法也如出一辙的，唯一不同的是，这些代码为性能带来了显著的提升，我们可以利用PyRex这样的工具允计C和Python混合编程，使编写扩展轻而易举，因为它会把所有的代码都转换成C语言代码。

因为Python的标准实现是使用C语言完成的（也就是CPython）,所以要使用C和C++编写Python扩展。Python的java实现被称为Jython，要使用java编写其扩展。最后还有IronPython这是针对.net平台的实现。

在各种不同的系统上你都可以看到Python的身影，因为Python是用C写的，又由于C的可移植性，使得Python可以运行在任何带有ANSI C编译平台上。

* 内置的数据类型

Python提供了一些内置的数据结构，这些数据结构实现了类似Java中集合类的功能，Python的数据结构包括元组、列表、字典等 。内置的数据结构简化了程序的设计。元组相当于“只读”的数组，列表可以作为可变长度的数组使用，字典相当于java中的HashTable类型。

* 健壮性

Python提供了异常处理机制，能捕获程序的异常情况。此外，Python的堆栈跟踪对象能够指出程序出错的位置和出错的原因。异常机制能够避免不安全退出的情况，同时能帮助程序员调试程序。

* 跨平台性

Python会先编译与平台相关的二进制代码，然后再解释执行，这种方式和Java相似。Python可以运行在Windows/Linux/MAC/Unix上

* 可扩展性

Python是采用C开发的语言，因此可以使用C扩展Python，可以给Python添加新的模块、新的类。

* 动态性

Python与Javascript、PHP、Perl等语言类似。Python不需要声明变量，直接赋予值可创建一个新的变量。

* 强类型语言

Python的变量创建后会对应一种数据类型，Python会根据赋值表达式的内容决定变量的数据类型。Python在内部建立了管理这些变量的机制，出现在同一个表达式中的不同类型的变量需要做类型转换。

* 应用广泛

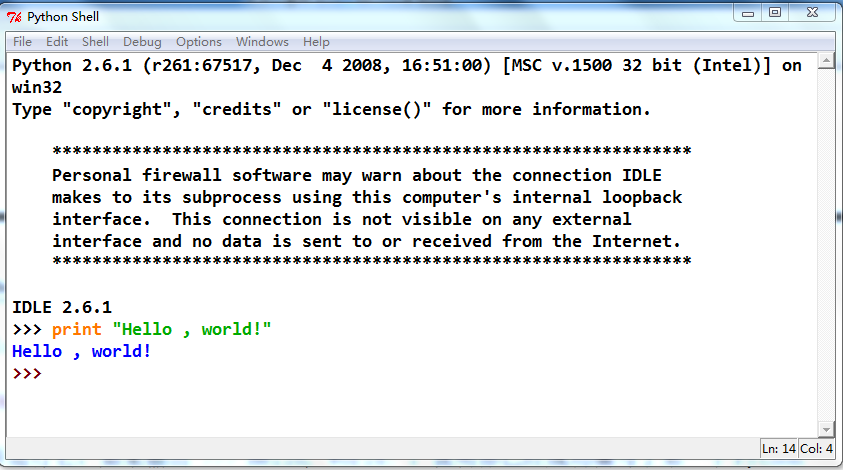
Python语言应用于数据库、网络、图形图像、数学计算、WEB开发、操作系统扩展等领域。Python有许多第三方库的支持。例如，PIL库用于图像处理、NumPy库用于数据计算、WxPython用于GUI库的设计、Django库用于WEB应用程序的开发等

## 第一个Python程序

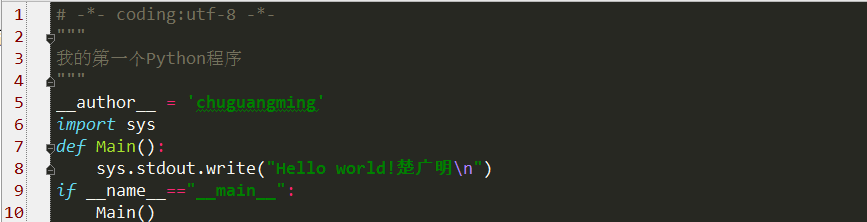
在Python的官方网站可以下载到Windows下的安装包，按照提示一路下去就可以了，记得要将Python所在的目录加入到系统Path变量中。

Python的安装包自带了一个简单的集成开发环境IDIE，你也可以选一个自己喜欢的IDE，我个人推荐PythonWin,它的语法提示功能不错，适合初学者使用。

现在你可以打开IDIE新建一个py为扩展名的Python脚本文件，输入以下内容：



保存并运行它，如果输出>>>Hello ,World!，这说明你已经成功编写了第一个Python程序，恭喜你！为了比较Python与Java编码风格等方面的差异，下边给出一个稍微复杂些的”Hello world”程序以及它的Java对照版本。



**请注意第一行代码是为了支持中文，没有这一行代码的话，我们在下面如果输出中文就会出错的。**

**Java的实现**

/\*\*

\* **@author** ChuguangMing

\*

\*/

**public** **class** myfirstjava

{

**public** **static** **void** main(String[] args)

{

System.*out*.println("这是Java的第一个程序");

}

}

**Python的实现**

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

**import** sys

**def** Main**():**

sys**.**stdout**.**write**(**"Hello World我的第一个程序\n"**)**

#下面的语句看起来比较奇怪，一会儿我们会解析它

**if** \_\_name\_\_**==**"\_\_main\_\_"**:**

Main**()**

### 为什么要学习Python

我学习过很多语言，C#、Java、C/C++、PHP，但是效高即高而且简单易懂的也就数Python了，可以说它是一种胶水语言，我们可以通过它将许多语言整合在一起使用。

我们来看一段代码：

#-\*-coding:utf-8-\*-

"""

my fist App

"""

import sys

import urllib

def Main():

htmlresult=urllib.urlopen("http://www.baidu.com").read()

print htmlresult

#this is test

if \_\_name\_\_=="\_\_main\_\_":

Main()

### 代码块与缩进的差异

**Java使用C/C++风格的编码形式，除了要求用{}组织代码块外，语句间的缩进可以是任意的。**

Python强制所有程序都有相同的编码风格，它通过缩进来组织代码块，缩进相同的语句被认为是处于同一个代码块中，在if/else等语句及函数定义式末尾会有一个冒号，指示代码块的开始。Python这种强制缩进的做法可以省去{}或者begin/end等，使程序的结构更为清晰（有的人认为恰好相反），同时也减少了无效的代码行数。

此外需要注意，尽量使用4个空格作为Python代码的一个缩进单位，最好不要使有TAB，更不要混用Tab和空格，这也算是Python的一个非强制性约定吧。

### 语句结尾的差异

Java用分号结尾，Python不用任何符号（类似BASIC）。实际上Python也可以使用分号结尾，像这样a=1;b=2;c=3;print a,b,c不过Python中这种风格多用于调试，应为你可以很容易注释掉这一行就删除了所有的调试代码。

另外，当一行很长时，Python可以用\符号折行显示代码。

### 注释方法

java//用单行注释,用/\*\*/进行多行注释，而Python用#符号进行单行注释，用三引号（可单可双）进行多行注释。

java的条件表达式必须要加括号，而Python的条件表达式加不加括号均可。

### 入口方法

java语言必须要有入口方法Main(),这是程序开始执行的地方。Python语言中没有入口方法（函数），作为解释型的语言，Python代码会自动从头执行。

如果你对这点不习惯，可以使用Python代码的内置属性\_\_name\_\_此属性会根据Python代码的运行条件变化：当Python代码以单个文件运行时，\_\_name\_\_便等于\_\_main\_\_，当你以模块形式导入使用Python代码时，\_\_name\_\_属性便是这个模块的名字。

当然，Python中的\_\_name\_\_属于并不是为了照顾C/C++/C#程序员的编程习惯而准备的，它主要目的是用于模块测试。想像一下在C#中编写一个组件或类代码时，一般还得同时编写一个调用程序来测试它。而Python中可以把二者合二为一，这就是\_\_name\_\_属性的真正作用。

### import与using语句

在用Python写代码时，我们首先import sys，这是导入了Python的sys模块，然后在代码里我们可以引用sys模块中的对象stdout及它的write方法。在Python中这是必须的，否则你无法调用sys模块中的任何东西。

简单的说，Python中的import相当于java中的包引用。最后import可以出现在代码的任何位置，只要在引用它之前出现就可以了，不过为了提高程序可读性，建议还是在所有代码开头书写import

## 小结

1. Python使用强制缩进的编码风格，并以此组织代码块
2. Python语句结尾不用分号
3. Python标明注释用#(单行)或三引号（多行）
4. Python语言没有入口方法（Main）,代码会从头到尾顺序执行
5. Python用import引用所需要的模块

# 基本数据类型

## 第一个程序的解析

“一切数据是对象，一切命名是引用”

如果你能理解这句话，说明对Python的变量与数据类型已经有了不错的认识，与Java不同，Python在使用变量之前无须定义它的类型，试着运行下面的例子：

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序")

i=1

print i

#下面的语句看起来比较奇怪，一会儿我们会解析它

if \_\_name\_\_=="\_\_main\_\_":

Main()

从上边我们可以看到，变量i在使用前并不需要定义，但是**必须声明以及初始化**该变量。试着运行下面的例子：

i=1

print i+j

上面的代码会产生一个异常：”NameError:name ‘h’ is not defined”，Python提示变量j没有定义。这点和BASIC等弱类型的语言不一样。

另一方面，Python与java有一个很大的差异就是在程序运行过程中，**同一变量名可以（在不同阶段）代表不同类型的数据**，看看下面的例子：

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序\n")

i=1

print i,type(i),id(i)

i=1000000000

print i,type(i),id(i)

i=1.1

print i,type(i),id(i)

#下面的语句看起来比较奇怪，一会儿我们会解析它

if \_\_name\_\_=="\_\_main\_\_":

Main()

变量i的类型在程序执行过程中分别经历了int long float的变化，这和静态类型语言如C语言有很大不同。静态语言只要有一个变量获取了一个数据类型，它就会一直是这个类型，变量名代表的是用来存放数据的内存位置。而Python中使用的变量名只是各种数据及对象的引用，用**id()**获取的才是存放数据的内存位置，我们输入的1、1000000000、1.1三个数据均会保存在**id()**所指示的这些内存位置中，直到垃圾回收把它们拉走，这是动态语言的典型特征，它确定一个变量的类型是在给它赋值的时候。

另一方面，Python又是强类型的，试着运行下边的例子：

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序\n")

i=10

j='虎虎'

print i+j

#下面的语句看起来比较奇怪，一会儿我们会解析它

if \_\_name\_\_=="\_\_main\_\_":

Main()

这个程序会出错，产生一个异常”TypeError:unsupported operand type(s) for +:’int’ and ‘str’”。一个正确的写法是：

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序\n")

i=10

j='虎虎'

print str(i)+j

#下面的语句看起来比较奇怪，一会儿我们会解析它

if \_\_name\_\_=="\_\_main\_\_":

Main()

所以，我们说Python即是一种动态类型语言，同时也是一种强类型的语言，这点是和java不同的地方，对于Python的这种变量的声明、定义、使用方式。Java程序员可能要花一段时间去适应，不过相信你会很快喜欢上它，因为它让事情变的更加简单（而且不会不安全）。

我们再小试牛刀一下，比如我们想看看c盘下面有什么数据与目录的话：

#-\*-coding:utf-8-\*-

import sys

import os

def main():

sys.stdout.write("Hello World我的第一个程序\n")

print os.listdir("c:/")

if \_\_name\_\_=="\_\_main\_\_":

main()

## 变量的命名规范

Python与java的变量（函数、类等其他标识符）的命名规范基本一样，同样对大小敏感。不一样的地方是Python中以下划线开始或者结束的标识符通常有特殊的意义。例如以一个下划线开始的标识符如**\_foo**不能用from module import \*语句导入。前名均有两个下划线的标识符，如**\_\_init\_\_**被特殊方法保留。前边有两个下划线的标识符，如**\_\_bar**，被用来实现类私有属性，这个将在“类和面对对象编程”中再说。

最后，Python的关键字不能作为标识符，不过Python的关键字比java要少得多，可以google一下，这里就列出了。

Python没有常量，如果你非要定义常量，可以引用const模块

Python程序中一切数据都是对象，包括自定义对象及基本数据类型，这一点和C#一样，它们都是完全面向对象的语言，所以我想C#程序员很容易理解**Python的一切数据都是对象**这个口号。

Python不区分值类型和引用类型，你可以把所有的类型都理解为引用类型（当然，它们的实现方式不是一样的，这里只是一个类比）。

Python内建的数据类型有20多种，其中有些不常用到的，有些即将合并。本文将主要介绍空类型、布尔类型、整型、浮点型和字符串、元组、列表、集合、字典等9种Python内置的数据类型。

在这里，我们将前4种称为“简单数据类型”，将后5种称为“高级数据类型”，实际上Python语言本身没有这种叫法，这样分类是我自已设定的，主要是为了和java的相关知识对照方便，希望不要误导大家。

## 空类型

空类型(None)表示该值是一个空对象，比如没有明确定义返回值的函数就是返回None。空类型没有任何属性，经常被用做函数中可选参数的默认值。None的布尔值为假。

Python的None和java中的可空类型Nullable<T>类似，比如java可以定义Nullable<double> i=null,与Python的空类型类似，但实现原理和用途都不一样。

## 布尔类型

Python中用True和False来定义真假，你可以直接用a=True或a=False来定义一个布尔型变量，但在Python2.6里,True/False以及None却都不是关键字，在Python3.0里它们已经是关键字了，这个有点乱， 我们可以不用管它，直接使用就OK了。

**注意和java不同的是，Python中True和False的首字母要大写。**

最后一点，在java中布尔类型和其他类型之间不存在标准的转换。但是在Python中，None、任何数值类型中的0、空字符串’’,空元组（），空列表[]，空字典{}都被当作False，其他对象均为True，这点和C++差不多，要提起注意，请思考一下，下面的Python代码会输出什么？

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序\n")

if 0:

print 'True'

else:

print 'False'

#下面的语句看起来比较奇怪，一会儿我们会解析它

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 数值类型

Python拥有四种数值类型：整型、长整型、浮点类型以及复数类型。

整数类型（int）用来表示从-2147483648到2147483647之间的任意整数（在某些电脑系统上这个范围可能会更大，但绝不会比这个更小）;长整型（long）可以表示任意范围的整数，**实际上我们把Python的long和int理解为同一种类型就可以了，因为当一个整数超过int的范围后，Python会自动将其升级为长整型**。

Python中只有64位双精度浮点数float，与java中的double类型相同(注意在Python中浮点数类型名字是float而不是double)，Python不支持32位单精度的浮点数。

除了整数和实数，Python还提供了一种特殊类型复数(complex)。复数使用一对浮点数表示，复数z的实部和虚部分分别用z.real和z.imag访问。

在数值运算中，整数与浮点数运算的结果是浮点数，这就是所谓的“提升规则”，也就是“小”类型会被提升为“大”类型参与计算。这一点和java是一样的，**提升的顺序依次为：int long float complex**

作为数值类型的最后一个问题，java程序员需要注意的是,**Python没有内建的decimal类型**，但可以导入decimal模块用来完成与货币相关的计算。

Python中序列是由非负整数索引的对象的有序集合，它包括字符串、Unicode字符串，列表、元组、xrange对象以及缓冲区对象。

## 字符串类型

Python拥有两种字符串类型：标准字符串(str)是单字节字符序列，Unicode字符串(unicode)是双字节字符序列。

在Python中定义一个标准字符串(str)可以使用单引号、双引号、三引号，这使得Python输入文本比java更方便。比如当Str的内容中包含双引号时，就可以用单引号定义，反之亦然。当字符中有换行符等特殊字符时，可以直接使用三引号定义。这样就方便了很多不用去记那些乱七八糟的转义字符。当然Python也支持转义字符，且含义与java基本一样。

下面是一个例子来说明这一点

#-\*-coding:utf-8-\*-

"""

我的第一个应用程序

"""

import sys

def Main():

sys.stdout.write("开始程序\n")

str1='i am "python"\n'

str2="i am 'Python' \r"

str3="""

i'm "Python",

<a href="http://www.sina.com.cn"></a>

"""

#你可以把html之类的东西都直接弄进来而不需要处理

print str1,str2,str3

if \_\_name\_\_=="\_\_main\_\_":

Main()

在Python中定义一个Unicode字符串，需要在引号前面加上一个字符u,例如

#-\*-coding:utf-8-\*-

import sys

def Main():

print u"我是派森"

if \_\_name\_\_=="\_\_main\_\_":

Main()

同时注意，当使用UTF-8编码时，非unicode字符中一个汉字的长度是3，而使用gb2312时是2，见下边代码：

#-\*-coding:utf-8-\*-

'''

test

'''

import sys

def Main():

print u"我是派森"

unicode =u'我'

str='我'

print len(unicode),len(str)

#输出的 1 3

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 全局变量

不加入global关键字的话，如果你是在函数内部定义的变量的话，在外部是无法获取的。

#coding:utf-8

def testList():

global list

list=[]

list.append("test1")

list.append("test2")

print list

testList()

print list

# 基本数据类型-列表(list【】)

## 概述

Python中列表（list）类似于java中的ArrayList,用于顺序存储结构。列表用符号[]表示，中间的元素可以是任何类型（包括列表本身，以实现多维数组），元素之间用逗号分隔。取值或赋值的时候可以像C数组一样，按位置索引：

#-\*-coding:utf-8-\*-

import sys

def Main():

array=[1,2,3]

print array[0]

#输出1

array[0]='a'

print array

#输出['a',2,3]

L=[123,'spam',1.23]

#输出大小

print len(L)

print L[0]

print L[:-1]#不包含最后一个

print L+[4,5,6]#重新拼接一个新的列表

if \_\_name\_\_=="\_\_main\_\_":

Main()

从上边的代码中你可能发现一个有趣的事情：在Python的列表中可以混合使用不同类型的数据，像[‘a’,2,3]这样，不过我不建议你这样做，我觉着没有什么好处。

另外还可以看到，列表是可变的序列，也就是说我们可以在“原地”改变列表上某个位置所存储的对象的值。

Python中的list支持多数的操作，同时list也支持“切片”这样的操作。切片指的是抽取序列的一部分，其形式为：list[start:end:step].其抽取规则是：从start开始，每次加上step，直到end为止。默认的step为1;当start没有给出时，默认从list的第一个元素开始;当end=-1时表示list的最后一个元素，依次类推。一些简单的例子见下边代码：

#-\*-coding:utf-8-\*-

import sys

def Main():

test=['never',1,2,'yes',1,'no','maybe']

print test[0:3]#包括test[0],不包括test[3]

print test[0:6:2]#包括test[0],不包括test[6],而且步长为2

print test[:-1]#包括开始，不包括最后一个

print test[-3:]#抽取最后3个

if \_\_name\_\_=="\_\_main\_\_":

Main()

字符串、列表、元组都支持切片操作，这个很方便，应该学会熟练使用它。

最后，list是Python中最基础的数据结构，你可以把它当作链表、堆栈或队列来使用，效率还不错。Python中没有固定长度数组，如果你确实很在意性能，可以改入array模块来创建一个C风格的数组，它的效率很高，这里就不详细介绍了。

我们还可以对其进行排序与反转

#-\*-coding:utf-8-\*-

import sys

def Main():

array=[5,2,3,1,8]

array.sort()

for s in array:

print s

array.reverse()

for s in array:

print s

if \_\_name\_\_=="\_\_main\_\_":

Main()

Python核心数据类型的一个优秀的特性就是它们支持任意的嵌套。能够以任意的组合对其进行嵌套。这种特性的一个直接应用就是实现矩阵，或者Python中的多维数组。一个嵌套列表的殂表能够完成这个基本的操作。

#-\*-coding:utf-8-\*-

import sys

def Main():

M=[[1,2,3],

[4,5,6],

[7,8,9]]

print M[0]

print M[1]

print M[2]

if \_\_name\_\_=="\_\_main\_\_":

Main()

处理序列的操作和列表的方法中，Python还包括了一个更高级的操作，称作列表解析表达式，从而提供了一种处理像矩阵这样结构的强大工具。列如，假设我们需要从列举的矩阵中提取出第二列。因为矩阵是按照行进行存储的，所以可以通过简单的索引即可获得行，使用列表解析可以同样简单地获得列。

#-\*-coding:utf-8-\*-

import sys

def Main():

M=[[1,2,3],

[4,5,6],

[7,8,9]]

col2=[row[1] for row in M]

print col2

col3=[row[1]+1 for row in M]

print col3

colfilter=[row[1] for row in M if row[1]%2==0]

print colfilter

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 常用方法

len(list)长度

del list 删除对象

列表对象支持的方法：

append(x) 尾部追加 单个对象x，使用多个对象会引起异常。

count(x) 返回对象x在list中出现的次数

extend(L) 将列表L中的项添加到表中

index(x) 返回匹配对象x第一个表项的索引，无匹配时产生异常

insert(i,x) 在索引‘i’的元素钱插入对象x

pop(x) 删除列表中索引x的表项，并返回同该表项的值，无参数删除最后

remove(x) 删除表匹配对象x的第一个元素，无匹配时异常

reverse() 颠倒列表元素的顺序

sort() 对列表排序

此外可简单使用+实现列表连接：[3,4] + [[1,2],5,6] --> [3,4,[1,2],5,6]

删除列表中的重复项：M = list(set(L))，python的set和其他语言类似, 是一个无序不重复元素集

# 基本数据类型-字典(dict｛｝)

## 概述

用过java中的字典的人对Hashtable应该不会陌生，Python里的哈希表就是字典(dict)了。与set类似，字典是一种无序存储结构，它包括关键字（key）和关键字对应的值(value)。

java程序员需要了解的就是,在Python中dict是一种内置的数据类型，定义方式为：

dictionary={key:value}#当有多个键值时，用逗号进行分割。

字典里的关键字为不可变类型，如字符串、整数、只包含不可变对象的元组，列表等不能作为关键字。字典中一个键只能与一个值关联，对于同一个键，后添加的值会覆盖之前的值。

学过数据结构的人对字典的散列查找效率应该都有认识，所以我建议在可能的情况下尽量多用字典，其它的就不多写了。

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

dict={"a":"apple","b":"banana","g":"grape","o":"orange"}

print dict

print dict["a"]

dict2={1:"apple",2:"banana",3:"grape",4:"orange"}

print dict2

print dict2[1]

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#字典的添加、删除、修改操作

dict={"a":"apple","b":"banana","g":"grape","o":"orange"}

dict["w"]="watermelon"

print dict

del(dict["a"])

print dict

print dict.pop("b")

#dict.clear()

#print dict

#字典的遍历

for k in dict:

print "dict[%s]="%k,dict[k]

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#字典的keys()与values()方法

dict={"a":"apple","b":"banana","g":"grape","o":"orange"}

#输出key的列表

print dict.keys()

#输出values的列表

print dict.values()

#-\*-coding:utf-8-\*-

import sys

def Main():

D={'food':'spam','quantity':4,'color':'pink'}

print D['food']

D['quantity']+=1

print D

#另外一种定义字典的方法

D={}

D['name']='Bob'

D['job']='dev'

D['age']=40

print D

#使用键值,进行排序

D={'a':1,'b':2,'c':3}

print D

Ks=D.keys()

print Ks

Ks.sort()

print Ks

for key in Ks:

print key,'=>',D[key]

for key in sorted(D):

print key,'=>',D[key]

#迭代与优化

squares=[x \*\* 2 for x in [1,2,3,4,5]]

print squares

#与以下代码是等效的

squares=[]

for x in [1,2,3,4,5]:

squares.append(x\*\*2)

if \_\_name\_\_=="\_\_main\_\_":

Main()

在访问的时候，如果这个键值不存在的话，如果我们没有做任何判断的话，会出现错误，这个时候我们可以用以下代码来进行判断。

#-\*-coding:utf-8-\*-

import sys

def Main():

D={'food':'spam','quantity':4,'color':'pink'}

#测试不存在的键值

if not D.has\_key('f'):

print '不存在这个键值'

else:

print D['f']

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 常用方法

has\_keys(x) 若字典中有x返回true

keys() 返回键的列表

values() 返回值的列表

dict.items() 返回tuples的列表。每个tuple有字典的dict的键和相应的值组成

clear() 删除词典的所有条目

copy() 返回字典的高层结构的拷贝，但不复制嵌入结构，而复制那些结构的引用。

update(x) 用字典x中的键/值对更新字典的内容。

get(x[,y]) 返回键x。若未找到返回None

# 基本数据类型-元组(tuple（）)

元组与列表非常相似，它用()而不是[]括起来的序列。元组比列表的速度更快，但元组是一个不可变的序列，也就是与str一样，无法在原位改变它的值。除此之外，其他属性与列表基本一致。

元组是Python中内置的一种数据结构。元组由不同的元素组成，每个元素可以存储不同类型的数据，如字符串、数字甚至元组。元组是写保护的！即元组创建后不能再做任何修改操作，元组通常代表一行数据，而元组中的元素代表不同的数据项。

元组的创建

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

tuple\_name=("apple","banana","grape","orange")

print tuple\_name[0]

分片输出

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

tuple\_name=("apple","banana","grape","orange")

print tuple\_name[1]

#支持分片输出

print tuple\_name[-1]

print tuple\_name[-2]

print tuple\_name[1:3]

#我还可以在元组中包含自已

print '-----------------'

tuple=(('t1','t2'),('t3','t4'))

print tuple[0][0]

print tuple[1][0]

实现解包的功能

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

tuple\_name=("apple","banana","grape","orange")

a,b,c,d=tuple\_name

print a,b,c,d

元组定义的方法与列表类似，不过在定义只包含一个元素的元组时，注意在后边加一个逗号，请体会以下几句的差异：

#-\*-coding:utf-8-\*-

import sys

def Main():

test=[0] #列表可以这样定义

print type(test) #输出<type 'list'>

test=[0,] #也可以这样定义

print type(test) #输出<type 'list'>

test=(0,) #元组可以这样定义

print type(test) #输出<type 'tuple'>

test=(0) #但不能这样定义，Python会认为它是一个括号表达式

print type(test) #输出<type 'int'>

test=0, #也可以省略括号，但要注意与C的逗号表达式不同

print type(test) #输出<type 'tuple'>

#还可以简单的交换数据

a=1

b=2

a,b=b,a

print a,b

if \_\_name\_\_=="\_\_main\_\_":

Main()

以上这类语句在Python中被广泛应用于变量交换、函数传值等应用，因此Python的解析器在不断对其进行优化，现在已经具备了相当高的效率。所以以上代码在Python2.5以后的版本中，比tmp=a;a=b;b=tmp这种常规语句更快。

* Python是一种动态的强类型语言，在使用变量之前无须定义其类型，但是必须声明和初始化
* “一切命名是引用”，Python中变量名是对象的引用，同一变量名可以在程序运行的不同阶段代表不同类型的数据
* “一切数据都是对象”，Python的所有数据类型都是对象，相较java具有一致的使用方法
* 把问题想得更简单一点，Python的数值类型可以说只有两种：整形和浮点
* 多使用list/tuple/set/dict这几种很pythonic的数据类型，它们分别用[]/()/([])/{}定义

# 基本数据类型-集合(set)

**Python中的set和java中的集合不是一个概念，这是翻译的问题，Python中的集合是指无序的、不重复的元素集，类似数学中的集合概念，可对其进行交、并、差、补等逻辑运算。**

常见集合的语法为:s=set([‘a’,’b’,’c’])。不过set在Python3.0中发生了较大的变化，创建一个集合的语法变成了:S={1,2,3},用花括号的方法，与后边要提到的dict类似。

如果在set中传入重复元素，集合会自动将其合并。这个特性非常有用，比如去除列表里大量的重复的元素，用set解决效率很高。示例如下：

#-\*-coding:utf-8-\*-

import sys

def Main():

a=[133,224,2344,2243,22342,224,133,133,989]

b=set(a)

print b

if \_\_name\_\_=="\_\_main\_\_":

Main()

另一个例子，找出两个list里面相同的元素(集合求交，其它类推)，代码如下：

#-\*-coding:utf-8-\*-

import sys

def Main():

a=[133,224,2344,2243,22342,224,133,133,989]

b=set(a)

print b

a=["11","22","33"]

b=["11","33"]

c=set(a)&set(b)

print c

if \_\_name\_\_=="\_\_main\_\_":

Main()

想想你如果自已实现这个算法会怎么写？然后可以找两个大一点的列表，比比和set实现的效率，你就会有体会了，在以后的程序中多用set吧。

# 数据类型转换

## 概述

## 常用方法

str(x) 将对象x翻译为字符串

list(x) 将对象序列x作为列表返回。例如‘hello’返回['h','e','l','l','o'],将tuple转换为列表

tuple(x) 将对象序列x作为tuple返回

int(x) 将字符串和数字转换为整数,对浮点进行舍位而非舍入

long(x) 将字符串和数字转换为长整形

float(x) 将str和num转换为浮点对象

complex(x,y) 将x做实部，y做虚部创建复数

hex(x) 将整数或长整数转换为十六进制字符串

oct(x) 将整数或长整数转换为八进制字符串

ord(x) 返回字符x的ASCII值

chr(x) 返回ASCII码x代表的字符

min(x[,...]) 返回序列中最小的元素

max(x[,...]) 返回序列中最大的元素

# 运算符、表达式和流程控制

本章介绍Python的运算符、表达式、程序流程控制语句以及异常处理语句，在这方面，Python和java是非常类似的，我们仅需要注意它们之间的一些细微差异。另外，在本章我还会简要介绍Python语言中的两项有趣功能=列表内涵和动态表达式，虽然它们严格来说属于函数部分的内容，不过我觉得还是放在表达式一章比较合适。

无论使用什么语言，我们编写的大多数代码都包含表达式。一个表达式可以分解为运算符和操作数，运算符的功能是完成某件事，它们由一些数学运算符号或者其他特定的关键字表示，运算符需要数据来进行运算，这样的数据被称为操作数。例如，2+3是一个简单的表达式，其中+是运算符，2和3是操作数。

## 算术运算符与算术表达式

算术运算符是程序设计语言最基本的运算符。Python提供的算术运算符除了+、-、\*、/、%（求余）之外，还提供了两种java中没有提供的运算符：求幂(\*\*)和取整除(//).下面我们就通过一段代码来解析这两个算术运算符的功能。

#-\*-coding:utf-8-\*-

import sys

def Main():

x=3.3

y=2.2

a=x\*\*y

print a

#输出即3.3的2.2次幂

b=x//y

print b

#输出1.0取整除返回商的整数部分

c=x/y

print c

#输出1.5，注意体会普通除与取整除的区别

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 赋值运算符与赋值表达式

赋值就是给一个变量赋一个新值，除了简单的=赋值之外，Python和java都支持复合赋值，例如x+=5,等价于x=x+5.

Python不支持java中的自增和自减运算符，例如X++这种语句在Python中会被提示语法有错误。java程序员可能习惯了这种表达式，在Python中，请老老实实的写X+=1就是了。

Python的逻辑运算符与java有较大区别，Python用关键字and or not 代替了java语言中的逻辑运算符 && || !，此外Python中参与逻辑运算符的操作数不限于布尔类型，任何类型的值都可以参与逻辑运算中去。

用逻辑运算符将操作数或表达式连接起来就是逻辑表达式。与java一样，Python中逻辑表达式是短路执行的，也就是说只有需要时才会进行逻辑表达式右边值的计算，例如表达式a and b 只有当a为true时才计算b.思考一下,if(0 and 10/0):这条语句会引发除数为零的异常吗？

此外还要注意：在Python中，and和or所执行的逻辑运算并不返回布尔值，而是返回它们实际进行比较的值之一。下边是一个例子：

#-\*-coding:utf-8-\*-

import sys

def Main():

print 'a' and 'b'

print '' and 'b'

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 关系运算符与关系表达式

关系运算符实际上是逻辑运算的一种，关系表达式的返回值总是布尔值。Python中的比较操作符与java完全是一样的，包括== != > < >= <=共6种。

除了基本的变量比较外，Python的关系运算符还包括身份运算符is.在Python中，is用来检验两个对象在内存中是否指向同一个对象（还记得一切数据皆对象吗？一切命名皆引用吗）。

三元运算符

三元运算符是c/c++系语言所特有的一类运算符，例如，对表达式b?x:y，先计算条件b,然后进行判断，如果b的值为true，则计算并返回x的值，否则计算并返回y的值。

在Python中，提供了专门的逻辑分支表达式来模拟java系中的三元运算，我们也可以在一行语句中完成三元运算，例如

print ‘偶数’ if x%2==0 else ‘奇数’

# 流程控制

## 条件语句

Python用if,elif,else这三个关键字进行条件判断，与java唯一的区别就是用elif取代else if，少打两个字，其它都一样，此外别忘了在if等语句后加：号

如果一个流程控制下不做任何事情，记得写一句pass语句，不然Python会报错。例如：

if 0:

pass #这一句语没有什么意义

在Python中没有switch语句，你可以使用if..elif..else语句来完成同样的工作。如果你觉得繁琐，可以试试dict实现的方式，下边是一个例子，分别对比了两种实现方式。

#-\*-coding:utf-8-\*-

import sys

def Main():

#使用if替代

x='4'

print "OK"

if x=='1':

print 'one'

elif x=='2':

print 'two'

else:

print 'nothing!'

#使用dict

numtrans={

1:'one',

2:'two',

3:'three'

}

try:

print numtrans[x]

except KeyError:

print 'nothing!'

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 循环

Python支持两种循环语句while循环和for循环，不支持java中的do-while循环。在Python的while循环和Java基本一致，此处我们着重比较两种语言中的for循环的区别。

说的简单一点，python中的for语句相当于java中的foreach语句,它用于从集合对象（list/str/tuple等）中遍历数据。例如：

for I in [1,2,3,4,5]:

print i

for I in range(10):

print i

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

tuple=(("apple","banana"),("grape","orange"))

for i in range(50,100+1):

print i

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

for i in range(50,100+1):

print i

range(1,5) #代表从1到5(不包含5)

range(1,5,2) #代表从1到5，间隔2(不包含5)

range(5) #代表从0到5(不包含5)

\_\_author\_\_ = 'TenYear'

#-\*-coding:utf-8-\*-

"""

my fist App

"""

import sys

import urllib

def Main():

itemlist=[1,2,3,4,5,4]

for m in itemlist:

print m

#this is test

if \_\_name\_\_=="\_\_main\_\_":

Main()

# 异常

Python和java一样支持异常处理，利用try/except/finally结构，可以很方便的捕获异常，同时可以用raise语句手动抛出异常（上述四个异常处理的关键字分别对就应Java中的try/catch/finally/throw）.通过except，您可以将try标示的语句中出现的错误和异常捕获。

#-\*-coding:utf-8-\*-

import sys

def Main():

try:

f=open('firstpython.py')

s=f.readline()

print s

except IOError,(errno,strerror):

print "I/O error(%s):%s" %(errno,strerror)

except ValueError:

print "Could not convert data to an integer"

except:

print "Unexpected error:",sys.exc\_info()[0]

raise

finally:

f.close()

if \_\_name\_\_=="\_\_main\_\_":

Main()

最后说明一点，python的try也支持else语句，如果有一些代码要在try没有发生异常的情况下才执行，就可以把它放在else中。

# 动态表达式

在C#语言中，如果需要在文本框中输入1+2（或更加复杂的数学表达式）后计算它的值，可能会用到表达式解析等。现在我们有了Python，要完成这种任务可以说是非常简单：只要用内置的eval()函数，就可以计算并返回任意有效表达式的值。例如：

str=’1+2’

print eval(str)

除了eval函数之外，Python还提供了exec语句将字符串str当成有效Python代码来执行，看下面的例子：

exec ‘a=100’

print a

另外还有execfile函数，它用来执行一个外部的py文件，上一个例子存为exec.py后，运行下边的代码就知道是怎么回事了：

execfile(r’c:\exec.py’)

最后提醒，默认的eval(),exec,execfile()所运行的代码都位于当前的名字空间中，eval()，exec,和execfile()函数也可以接受一个或两个可选字典参数作为代码执行的全局名字空间和局部名字空间，具体可以参考Python手册。

列表内涵是Python最强有力的语法之一，常用于从集合对象中有选择地获取并计算元素，虽然多数情况下可以使用for/if等语句组合完成同样的任务，但列表内涵书写的代码更加简洁。

#-\*-coding:utf-8-\*-

import sys

def func():

x=1

y=2

m=3

n=4

sum=lambda x,y:x+y

print sum

sub=lambda m,n:m-n

print sub

return sum(x,y)\*sub(m,n)

if \_\_name\_\_=="\_\_main\_\_":

print func()

列表内涵的一般形式如下，我们可以把[]内的列表内涵写为一行，也可以写为多行。

[表达式 for item1 in 序列1 … for itemN in 序列N if 条件表达式]

上面的表达式分为三部分，最左边是生成每个元素的表达式，然后是for迭代过程，最右边可以设定一个if判断作为过滤条件。

列表内涵的一个著名例子是生成九九乘法表：

s=[(x,y,x\*y) for x in range(1,10) for y in range(1,10) if x>=y]

# 函数及函数编程

在java中没有独立的函数存在，只有类的（动态或静态）方法这一概念，它指的是类中用于执行计算或其它行为的成员。在Python中，你可以使用类似C#的方式定义类的动态或静态成员方法，因为它与java一样支持完全的面向对象编程。你也可以用过程式编程的方式来编写Python程序，这时Python中的函数与类可以没有任何关系，类似C语言定义和使用函数的方式。此外，Python还支持函数式编程，虽然它对函数编程的支持不如LISP等语言那样完备，但适合使用还是可以提高我们工作的效率的。

## 函数的定义

函数定义是最基本的行为抽象代码，也是软件复用最初级的方式。Python中函数的定义语句由def关键字、函数名、括号、参数及冒号组成。下面是几个简单的函数定义语句：

#-\*-coding:utf-8-\*-

import sys

def Main():

def F1():

print "我是F1"

def F2(x,y):

a=x+y

return a

#定义有多个返回值的函数，用逗号分割不同的返回值，

#返回结果是一个元组

def F3(x,y):

a=x/y

b=x%y

return a,b

if \_\_name\_\_=="\_\_main\_\_":

Main()

可能你已经注意到了，Python定义函数的时候并没有约束参数的类型，它以最简单的形式支持了泛型编程。你可以输入任意类型的数据作为参数，只要这些类型支持函数内部的操作

## 定义一个交换函数

#-\*-coding:utf-8-\*-

import sys

import random

def compareNum(num1,num2):

if(num1>num2):

return 1

elif(num1==num2):

return 0

else:

return -1

if \_\_name\_\_=="\_\_main\_\_":

num1=random.randrange(1,9)

num2=random.randrange(1,9)

print "num1=",num1

print "num2=",num2

print compareNum(num1,num2)

## 函数的默认参数与返回值

#-\*-coding:utf-8-\*-

import sys

def arithmetic(x=1,y=1,operator="+"):

result={

"+":x+y,

"-":x-y,

"\*":x\*y,

"/":x/y

}

return result.get(operator)

if \_\_name\_\_=="\_\_main\_\_":

print arithmetic(1,2)

print arithmetic(1,2,"-")

## 返回多个值

Python支持多个返回值

#coding:utf-8

def testList():

return "aaa","bbb"

a,b=testList()

print a

print b

## locals函数和globals函数介绍

Python有两个内置的函数，locals() 和globals()，它们提供了基于字典的访问局部和全局变量的方式。首先，是关于名字空间的一个名词解释。是枯燥，但是很重要，所以要耐心些。Python使用叫做名字空间的东西来记录变量的轨迹。名字空间只是一个字典，它的键字就是变量名，字典的值就是那些变量的值。实际上，名字空间可以象Python的字典一样进行访问，一会我们就会看到。

在一个Python程序中的任何一个地方，都存在几个可用的名字空间。每个函数都有着自已的名字空间，叫做局部名字空间，它记录了函数的变量，包括 函数的参数和局部定义的变量。每个模块拥有它自已的名字空间，叫做全局名字空间，它记录了模块的变量，包括函数、类、其它导入的模块、模块级的变量和常量。还有就是内置名字空间，任何模块均可访问它，它存放着内置的函数和异常。

当一行代码要使用变量 x 的值时，Python会到所有可用的名字空间去查找变量，按照如下顺序：

1.局部名字空间 - 特指当前函数或类的方法。如果函数定义了一个局部变量 x，Python将使用这个变量，然后停止搜索。

2.全局名字空间 - 特指当前的模块。如果模块定义了一个名为 x 的变量，函数或类，Python将使用这个变量然后停止搜索。

3.内置名字空间 - 对每个模块都是全局的。作为最后的尝试，Python将假设 x 是内置函数或变量。

如果Python在这些名字空间找不到 x，它将放弃查找并引发一个 NameError 的异常，同时传递 There is no variable named 'x' 这样一条信息,象Python中的许多事情一样，名字空间在运行时直接可以访问。特别地，局部名字空间可以通过内置的 locals 函数来访问。全局（模块级别）名字空间可以通过 globals 函数来访问

#coding:utf-8

**from** \_\_future\_\_ **import** unicode\_literals

**from** sys **import** **\***

**def** test**(**arg**):**

#函数test在它的局部名字空间中有两个变量:arg(它的值被传入函数)

# 和z(它在函数内部)

z**=**1

**print** locals**()**

#locals返回一个名字与值的字典，这个字典的键字是字符串形式的变量名

#字典的值是变量的实际值

test**(**5**)**

test**(**'楚广明'**)**

**print** globals**()**#返回一个全局的字典，包括所有导入的变量

# 类及面向对象

如果你熟悉java或者c#,那么对类和面向对象应该不会陌生。Python与c#一样，能够很好的支持面向对象的编程模式。

## 类的定义

与C#一样，Python使用class关键字定义一个类。一个最简单的类定义语句如下：

Class A:

Pass

它等价于C#中的class A{}.当然以上语句没有任何实际意义，它只是告诉我们什么是定义一个类所必需要的。即class关键字，类名和冒号，**pass关键字只用来占位，相当于用来占位，相当于C#中花括号的作用。**

类是定义对象格式的模板，而对象则是类的实例，通过类创建对象的过程称为类的实例化。在C#中，需要使用new关键字实例化一个类，例如

A a=new A();

在上条语句，C#完成了两件事，首先声明一个类型为A的变量a,然后用new运算符创建一个类型为A的对象，并将该对象的引用赋值给变量a,而在python中没有new 关键字，同时它是一个动态语言，不需要事先指定变量的类型，只需要：

a=A()

即创建一个类型为A的对象，看起来好像是将类当作一个函数调用，返回值是新创建的对象。

## 为类添加数据

通常我们利用类来定义各种新的数据类型，其中即包含数据内容，也包含对数据内容的操作。Python类的数据添加方法与C#有一些不同，因为Python是一种动态语言，变量在使用之前不需要定 义，所以你可以不在类定义中添加成员变量，而是在运行时动态地添加它们，例如：

#-\*-coding:utf-8-\*-

import sys

def Main():

class A:pass

a=A()

a.x=1

print a.x

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 构造函数

Python的类提供了类似C#构造函数的东西：\_\_init\_\_(**注意是前后是两个下划线**)，类在实例化时会首先调用这个函数，我们可以通过重写\_\_init\_\_函数，完成变量的初始化等工作。与C#不同的地方是，**Python不支持无参数的初始化函数，你至少需要为初始化函数指定一个参数，即对象实例本身(self)**.下面是一段简单的代码。在该代码中，我们重写了函数\_\_init\_\_，定义并初始化一个类的成员变量X：

#-\*-coding:utf-8-\*-

import sys

def Main():

class A:

def \_\_init\_\_(self):

self.x=1

a=A()

print a.x

if \_\_name\_\_=="\_\_main\_\_":

Main()

#-\*-coding:utf-8-\*-

import sys

class Student:

def \_\_init\_\_(self,name,age):

self.\_\_name=name

self.\_\_age=age

def getName(self):

format="my name is %s my age is %d"%(self.\_\_name,self.\_\_age)

print format

def \_\_del\_\_(self):

print "del"

if \_\_name\_\_=="\_\_main\_\_":

student=Student("chu",35)

student.getName()

## 类的访问修饰符

以前我们在使用C#或都java这一类语言时，经常使用public private之类的访问修饰符，但是在python中就大大的不一样了。请看以下二段代码的比较。

**namespace** ConsoleApplication1

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

A a **=** **new** A**();**

Console**.**WriteLine**(**A**.**y**);**

Console**.**WriteLine**(**a**.**x**);**

**}**

**}**

class A

**{**

**public** static int y **=** 2**;**

**public** int x**;**

**private** int z**;**

**public** A**()**

**{**

x **=** 1**;**

z **=** 1**;**

**}**

**}**

**}**

#-\*-coding:utf-8-\*-

import sys

class A:

#定义静态成员变量

y=2

def \_\_init\_\_(self):

#定义公共的成员变量

self.x=1

#定义私有的成员变量

self.\_\_z=1

if \_\_name\_\_=="\_\_main\_\_":

a=A()

#打印公共的成员变量

print a.x

#打印静态成员变量

print A.y

#打印私有成员变量出错

#print a.\_\_z

## 为类添加方法

方法是对类数据内容的操作，在Python中，定义类的方法与定义一个普通的函数在语法上基本上相同，c#程序员需要注意的是，在类中定义的常规方法的第一个参数总是该类的实例，即self，同时注意在方法中引用类的另一个方法必须使用类名加方法名的形式，下面是一个定义类的方法的简单的例子。

#-\*-coding:utf-8-\*-

import sys

class A:

def prt(self):

print "my name is A"

def reprt(self):

A.prt(self)

if \_\_name\_\_=="\_\_main\_\_":

a=A()

a.prt()

a.reprt()

不能定义一个不操作实例的方法

#-\*-coding:utf-8-\*-

import sys

class Student:

\_\_name=""

def \_\_init\_\_(self,name):

self.\_\_name=name

def getName(self):

return self.\_\_name

if \_\_name\_\_=="\_\_main\_\_":

student=Student("chu")

print student.getName()

## 静态方法

Python与C#一样支持静态方法。在C#中需要使用关键字static声明一个静态方法，而在Python中是通过静态方法修饰符@staticmethod来实现的，下面是例子：

#-\*-coding:utf-8-\*-

import sys

class A:

def prt(self):

print "my name is A"

def reprt(self):

A.prt(self)

@staticmethod

def prt2():

print "我是静态方法"

if \_\_name\_\_=="\_\_main\_\_":

a=A()

a.prt()

a.reprt()

A.prt2()

如你所见，静态方法可以直接被类调用，它没有常规方法那样的特殊行为（默认第一行参数是self等 ），你完全可以将静态方法当成一个用属性引用方式调用的普通函数。

## 单继承

Python用类名后加扩号的方式实现继承，下面是一个简单的示例：

#-\*-coding:utf-8-\*-

import sys

class A:

x=1

class B(A):

y=2

if \_\_name\_\_=="\_\_main\_\_":

print B.x

print B.y

一个复杂的实例

#-\*-coding:utf-8-\*-

import sys

class SchoolMember:

'''Represents any school member.'''

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

print'(Initialized SchoolMember: %s)'% self.name

def tell(self):

'''Tell my details.'''

print'Name:"%s" Age:"%s"'% (self.name, self.age),

class Teacher(SchoolMember):

'''Represents a teacher.'''

def \_\_init\_\_(self, name, age, salary):

SchoolMember.\_\_init\_\_(self, name, age)

self.salary = salary

print'(Initialized Teacher: %s)'% self.name

def tell(self):

SchoolMember.tell(self)

print'Salary: "%d"'% self.salary

class Student(SchoolMember):

'''Represents a student.'''

def \_\_init\_\_(self, name, age, marks):

SchoolMember.\_\_init\_\_(self, name, age)

self.marks = marks

print'(Initialized Student: %s)'% self.name

def tell(self):

SchoolMember.tell(self)

print'Marks: "%d"'% self.marks

if \_\_name\_\_=="\_\_main\_\_":

t = Teacher('Mrs. Shrividya',40,30000)

s = Student('Swaroop',22,75)

## 传值与传引用

1. python不允许程序员选择采用传值还是传引用。Python参数传递采用的肯定是“传对象引用”的方式。实际上，这种方式相当于传值和传引用的一种综合。如果函数收到的是一个可变对象（比如字典或者列表）的引用，就能修改对象的原始值——相当于通过“传引用”来传递对象。如果函数收到的是一个不可变对象（比如数字、字符或者元组）的引用，就不能直接修改原始对象——相当于通过“传值'来传递对象。

2. 当人们复制列表或字典时，就复制了对象列表的引用同，如果改变引用的值，则修改了原始的参数。

3. 为了简化内存管理，Python通过引用计数机制实现自动垃圾回收功能，Python中的每个对象都有一个引用计数，用来计数该对象在不同场所分别被引用了多少次。每当引用一次Python对象，相应的引用计数就增1，每当消毁一次Python对象，则相应的引用就减1，只有当引用计数为零时，才真正从内存中删除Python对象。

#-\*-coding:utf-8-\*-

**def** add\_list**(**p**):**

p**=**p**+[**1**]**

**def** add\_list2**(**p2**):**

p2**+=[**1**]**

tmpList**=[**1**,**2**,**3**,**4**]**

add\_list**(**tmpList**)**

**print** tmpList

**#结果[1,2,3,4]**

add\_list2**(**tmpList**)**

**print** tmpList

**#结果[1,2,3,4,1]**

这主要是由于“=”操作符会新建一个新的变量保存赋值结果，然后再把引用名指向“=”左边，即修改了原来的p引用，使p成为指向新赋值变量的引用。而+=不会，直接修改了原来p引用的内容，事实上+=和=在python内部使用了不同的实现函数。

#-\*-coding:utf-8-\*-

a**=[]**

b**={**'num'**:**0**,**'sqrt'**:**0**}**

resurse**=[**1**,**2**,**3**]**

**for** i **in** resurse**:**

b**[**'num'**]=**i

b**[**'sqrt'**]=**i**\***i

a**.**append**(**b**)**

**print** a

#结果[{'num': 3, 'sqrt': 9}, {'num': 3, 'sqrt': 9}, {'num': 3, 'sqrt': 9}]

#这是由于a中的元素就是b的引用。可以修改为：

#修改

a**=[]**

**for** i **in** resurse**:**

a**.**append**({**'num'**:**i**,**"sqrt"**:**i**\***i**})**

**print** a

# 模块与包

Python的脚本都是用扩展名为py的文本文件保存的，一个脚本可以单独运行，也可以导入另一个脚本中运行。当脚本被导入运行时，我们将其称为模块（module）。模块是Python组织代码的基本方式。

Python的程序是由包（package）、模块（module）和函数组成.包是由一系列模块组成的集合。模块是处理某一类问题的函数和类的集合。

包就是一个完成特定任务的工具箱，Python提供了许多有用的工具包，如字符串处理、图形用户接口、WEB接口、图形图像处理等。使用这些工具包，可以提高程序员的开发效率、减少编程的复杂度、达到代码重用的效果。这些自带的工具包和模块安装在Python的安装目录下的Lib子目录中。

例如，Lib目录中的xml 文件夹就是一个包，这个包用于完成XML的应用开发。Xml包中有几个子包：dom、sax、etree和parser。文件\_\_init\_\_.py（注意是两个下滑线）是xml包的注册文件，如果没有该文件，Python将不能识别xml包。在系统字典中定义了xml包。

包必须至少含有一个\_\_init\_\_.py文件，该文件的内容可以为空。\_\_init\_\_.py用于表示当前文件夹是一个包。

## 对于模块的理解

用简单的说法来说，每一个以扩展名为.py结尾的Python源代码都是一个模块。其他的文件可以通过导入一个模块读取这个模块的内容。导入从本质上来讲，就是载入另一个文件，并能够读取那个文件的内容。一个模块的内容通过这样的的属性能够被外部世界使用。

比如我们举一个简单的例子，首先建立一个python文件define.py

#-\*-coding:utf-8-\*-

myvar="这是一个测试"

很简单的代码就二行，我们定义了一个变量myvar.现在我们通过另一个python文件来导入它。

#-\*-coding:utf-8-\*-

import sys

import define

def Main():

print define.myvar

print dir(define)

print define.\_\_file\_\_

if \_\_name\_\_=="\_\_main\_\_":

Main()

import define #这一行语句的意义在于导入define.py这个模块

print define.myvar#这一行的意义在于打印myvar这个变量

print dir(define)#这一行的意义在于输出所有可以使用的变量

输出结果，不言而喻。但是在默认情况下，只是在每次会话的第一次运行。在第一次导入之后，其他的导入都不会再工作。

但是如果真的想要python在同一次会话中再次运行文件（不停止和重新启动会话），需要调用内置的reload函数。

## 模块的显要特性：属性

导入和重载提供了一种自然的程序启动的选择，因为导入操作将会在最后一步执行文件。从更宏观的角度来看，模块扮演了一个工具库的角色。我们可以直接使用from define import myvar这条语句来实现。

#-\*-coding:utf-8-\*-

import sys

from define import myvar

def Main():

print myvar

if \_\_name\_\_=="\_\_main\_\_":

Main()

## 模块创建过程的例子

模板把一组相关的函数或代码组织到一个文件中。一个文件即是一个模板。模块由代码、函数或类组成。例如，创建一个名为myModule.py的文件，即定义了一个名为myModule的模块。在该模块中定义一个函数func()和一个类MyClass。MyClass类中定义一个方法myFunc().

#-\*-coding:utf-8-\*-

import sys

def func():

print "myModule.func()"

class MyClass:

def myFunc(self):

print "myModule.MyClass.myFunc()"

然后在myModule.py所在目录下创建一个名为call\_myModule.py文件，在该文件中调用myModule模块的函数和类：

#-\*-coding:utf-8-\*-

import sys

import random

import myModule

if \_\_name\_\_=="\_\_main\_\_":

myModule.func()

myClass=myModule.MyClass()

myClass.myFunc()

当python寻入一个模块时，python首先查找当前路径，然后查找lib目录、site-packages目录(python/lib/site-packages)和环境变量PYTHONPATH设置的目录。

## 模块的导入

在使用一个模块中的函数或类之前，首先要导入该模块。

import myModule

还可以使用from …import..语句将模块导入。

from module\_name import \*

from module\_name import function\_name

实现例子

#-\*-coding:utf-8-\*-

import sys

def func():

print "myModule.func()"

class MyClass:

def myFunc(self):

print "myModule.MyClass.myFunc()"

#-\*-coding:utf-8-\*-

import sys

import random

from myModule import func

if \_\_name\_\_=="\_\_main\_\_":

func()

## 模块的属性

模块中有许多内置的属性，用于完成特定的任务，如\_\_name\_\_ \_\_doc\_\_\_。每个模块都有一个名称。

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

print \_\_name\_\_

print \_\_doc\_\_

print \_\_file\_\_

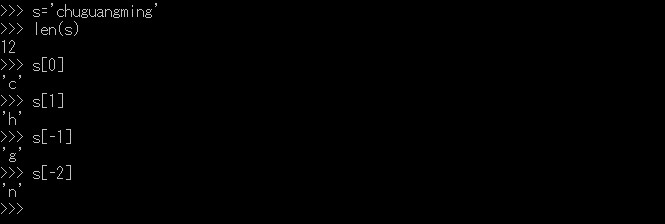
print \_\_package\_\_

# 字符串与简单的正则表达式

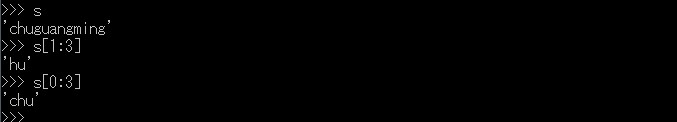
本章将介绍Python中字符串和正则表达式的概念。字符串是程序开发中常用的数据类型，字符串的处理是实际应用中实际应用中经常面对的问题。Python提供了功能强大的字符串模块，正则表达式专门用于匹配应用中的数据，能够简化字符串的处理程序，Python提供了re模块用来匹配正则表达式。

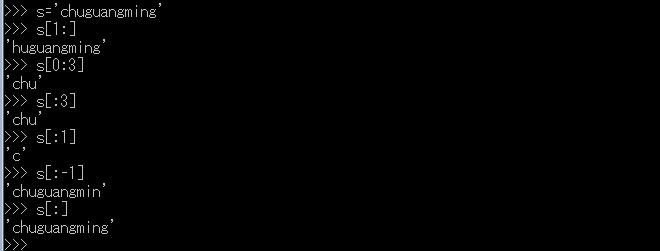
如果我们有一个含有12个字符的字符串，我们通过内置的len函数验证其长度并通过索引操作得到其各个元素。

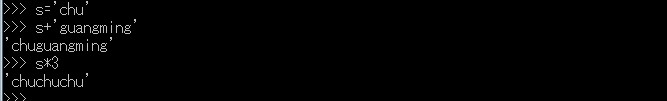
在Python中，索引是按照从最前面的偏移量进行编码的，也就是从0开始，第一项索引为0，第二项索引为1，依次类推。在python中，我们能够反向索引，从最后一个开始。



除了简单的从位置进行索引，序列也支持一种所谓分片的操作，这是一种一步就能够提取整个分片的方法。例如：







## 解决中文字符的问题

#-\*- coding:utf-8 -\*-

from \_\_future\_\_ import unicode\_literals

print(type("test")) # output: <type 'unicode'>

spam = "测试字符串"

print(spam[1:3]) # output:试字

## 单双引号的区别

先说1双引号与3个双引号的区别，双引号所表示的字符串通常要写成一行 如：

s1 = "hello,world"

如果要写成多行，那么就要使用\ (“连行符”)吧，如

s2 = "hello,\

world"

s2与s1是一样的。如果你用3个双引号的话，就可以直接写了，如下：

s3 = """hello,

world,

hahaha."""

那么s3实际上就是"hello,\nworld,\nhahaha.", 注意“\n”，所以， 如果你的字符串里\n很多，你又不想在字符串中用\n的话，那么就可以使用3个双引号。而且使用3个双引号还可以在字符串中增加注释，如下：

s3 = """hello, #hoho, this is hello, 在3个双引号的字符串内可以有注释哦

world, #hoho, this is world

hahaha."""

这就是3个双引号和1个双引号表示字符串的区别了，3个双引号与1个单引号的区别也是和这个一样的，实际上python支持单引号是有原因的，下面我来比较1个单引号和 1个双引号的区别。

当我用单引号来表示一个字符串时，如果要表示 Let's go 这个字符串，必须这样：

s4 = 'Let\'s go'，注意没有，字符串中有一个'，而字符串又是用'来表示，所以这个时候就要使用转义符 \ （\，转义符应该知道吧）, 如果你的字符串中有一大堆的转义符，看起来肯定不舒服，python也很好的解决了这个问题，如下：

s5 = "Let's go"

这时，我们看，python知道你是用 " 来表示字符串，所以python就把字符串中的那个单引号 ' , 当成普通的字符处理了，是不是很简单。 对于双引号，也是一样的，下面举个例子

s6 = 'I realy like "python"!'

这就是单引号和双引号都可以表示字符串的原因了。

## 字符串的格式化

C语言使用函数printf()、sprintf()格式化输出结果，Python也提供了类似的功能。Python将若干值值插入到带有%标记的字符串，从而可以动态的输出字符串。字符串的格式化语法如下所示：

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#格式化字符串

str1="version"

num=1.0

format="%s"%str1

print format

format="%s%d"%(str1,num)

print format



#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#带精度的格式化字符串

print "浮点型数字:%f"%1.25

print "浮点型数字:%.1f"%1.25

print "浮点型数字:%.2f"%1.254

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#带精度的格式化字符串

print "浮点型数字:%f"%1.25

print "浮点型数字:%.1f"%1.25

print "浮点型数字:%.2f"%1.254

#使用字典格式化字符串

print"%(version)s:%(num).1f"%{"version":"version","num":2}

#字符串对齐

word="version3.0"

print word.center(20)

print word.center(20,"\*")

#转义字符

path="hello\tworld\n"

print path

print len(path)

#直接输出转义

path=r"hello\tworld\n"

print path

## 字符串的合并与分割

与java语言一样，Python使用”+”号连接不同的字符串，Python会根据加号两侧变量的类型，还决定连接操作或加法运算。

#-\*-coding:utf-8-\*-

import sys

if \_\_name\_\_=="\_\_main\_\_":

#使用字符串截取子串

word="world Hello"

print word[0]

#string[start:end:step]

#从0开始截取三个

print word[0:3]

#从0开始

print word[0::2]

#使用split()获取子串

sentence="Bob Said:1,2,3,4"

print "使用空格取子串：",sentence.split()

print "使用逗号取子串：",sentence.split(",")

print "使用两个逗号取子串：",sentence.split(",",2)

#-\*- coding:utf-8-\*-

import sys

def Main():

print "string test example"

s='chuguangming'

#字符串替换

print s.find('chu')

print s.replace('chu','\*\*\*')

#字符串分割

line='aaa,BBB,ccc,ddd'

result= line.split(',')

for s in result:

print s

#大小写转换

line=line.upper()

print line

#判断是不是字符串

print line.isalnum()

print line.isalpha()

#去除空格

line='aaa,bbb,ccc,ddd\n'

print line.rstrip()

if \_\_name\_\_=="\_\_main\_\_":

Main()

正则表达式用于搜索、替换、解析字符串。正则表达式的功能强大，使用非常灵活。使用正则表达式需要遵守一定的语法规则，用来编写一些逻辑验证非常方便，例如电子邮件的验证。Python提供了re模块实现正则表达式的验证。

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#使用^与$的使用方法

s="HELLO WORLD"

#匹配hello开始的字符串，由于变量s中的HELLO是大写的所以匹配失败

print re.findall(r"^hello",s)

#re.I表示时忽略大小写

print re.findall(r"^hello",s,re.I)

#$的意思是匹配尾部

print re.findall(r"WORLD$",s)

#匹配每个英文单词\b用于分割单词

print re.findall(r"\b\w+\b",s)

## 常用字符操作演示

去空格及特殊符号

s.strip().lstrip().rstrip(',')

复制字符串

#strcpy(sStr1,sStr2)

sStr1 = 'strcpy'

sStr2 = sStr1

sStr1 = 'strcpy2'

print sStr2

连接字符串

#strcat(sStr1,sStr2)

sStr1 = 'strcat'

sStr2 = 'append'

sStr1 += sStr2

print sStr1

查找字符

#strchr(sStr1,sStr2)

# < 0 为未找到

sStr1 = 'strchr'

sStr2 = 's'

nPos = sStr1.index(sStr2)

print nPos

比较字符串

#strcmp(sStr1,sStr2)

sStr1 = 'strchr'

sStr2 = 'strch'

print cmp(sStr1,sStr2)

扫描字符串是否包含指定的字符

#strspn(sStr1,sStr2)

sStr1 = '12345678'

sStr2 = '456'

#sStr1 and chars both in sStr1 and sStr2

print len(sStr1 and sStr2)

字符串长度

#strlen(sStr1)

sStr1 = 'strlen'

print len(sStr1)

将字符串中的大小写转换

#strlwr(sStr1)

sStr1 = 'JCstrlwr'

sStr1 = sStr1.upper()

#sStr1 = sStr1.lower()

print sStr1

追加指定长度的字符串

#strncat(sStr1,sStr2,n)

sStr1 = '12345'

sStr2 = 'abcdef'

n = 3

sStr1 += sStr2[0:n]

print sStr1

字符串指定长度比较

#strncmp(sStr1,sStr2,n)

sStr1 = '12345'

sStr2 = '123bc'

n = 3

print cmp(sStr1[0:n],sStr2[0:n])

复制指定长度的字符

#strncpy(sStr1,sStr2,n)

sStr1 = ''

sStr2 = '12345'

n = 3

sStr1 = sStr2[0:n]

print sStr1

将字符串前n个字符替换为指定的字符

#strnset(sStr1,ch,n)

sStr1 = '12345'

ch = 'r'

n = 3

sStr1 = n \* ch + sStr1[3:]

print sStr1

扫描字符串

#strpbrk(sStr1,sStr2)

sStr1 = 'cekjgdklab'

sStr2 = 'gka'

nPos = -1

for c in sStr1:

if c in sStr2:

nPos = sStr1.index(c)

break

print nPos

翻转字符串

#strrev(sStr1)

sStr1 = 'abcdefg'

sStr1 = sStr1[::-1]

print sStr1

查找字符串

#strstr(sStr1,sStr2)

sStr1 = 'abcdefg'

sStr2 = 'cde'

print sStr1.find(sStr2)

分割字符串

#strtok(sStr1,sStr2)

sStr1 = 'ab,cde,fgh,ijk'

sStr2 = ','

sStr1 = sStr1[sStr1.find(sStr2) + 1:]

print sStr1

#或者

s = 'ab,cde,fgh,ijk'

print(s.split(','))

连接字符串

delimiter = ','

mylist = ['Brazil', 'Russia', 'India', 'China']

print delimiter.join(mylist)

PHP 中 addslashes 的实现

def addslashes(s):

d = {'"':'\\"', "'":"\\'", "\0":"\\\0", "\\":"\\\\"}

return ''.join(d.get(c, c) for c in s)

s = "John 'Johny' Doe (a.k.a. \"Super Joe\")\\\0"

print s

print addslashes(s)

只显示字母与数字

def OnlyCharNum(s,oth=''):

s2 = s.lower();

fomart = 'abcdefghijklmnopqrstuvwxyz0123456789'

for c in s2:

if not c in fomart:

s = s.replace(c,'');

return s;

print(OnlyStr("a000 aa-b"))

# 文件IO

## 基本文件功能演示

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#创建文件

context='''hello world

hello china '''

f=file('hello.txt','w')

f.write(context)

f.close()

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#读取文件

f=open("hello.txt")

while True:

line=f.readline()

if line:

print line

else:

break

f.close()

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#使用readlnes()读取多个文件

f=file("hello.txt")

lines=f.readlines()

print lines

for line in lines:

print line

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#一次性读取

f=open("hello.txt")

context=f.read()

print context

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

f=open("hello.txt")

context=f.read(5)

print context #读取前5个字节的数据

print f.tell()#显示当前的位置

context=f.read(5)

print context

print f.tell()

f.close()

## 文件的写入

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#使用writelines()写文件

f=file("hello.txt","w+")

li=["hello chu888\n","hello li\n"]

f.writelines(li)

f.close()

#-\*-coding:utf-8-\*-

import sys

import re

if \_\_name\_\_=="\_\_main\_\_":

#使用writelines()写文件

f=file("hello.txt","w+")

li=["hello chu888楚\n","hello li\n"]

f.writelines(li)

f.close()

#追加文件内容

f=file("hello.txt","a+")

new\_context="goodbye"

f.write(new\_context)

f.close()

## 文件的删除与复制

#-\*-coding:utf-8-\*-

import sys

import re

import os

if \_\_name\_\_=="\_\_main\_\_":

file("hello.txt","w")

if os.path.exists("hello.txt"):

os.remove("hello.txt")

file类没有提供文件拷贝的功能，但是我们可以使用read()、write()方法模拟实现文件的拷贝，但是最好的方法是引入shutil模块

#-\*-coding:utf-8-\*-

import sys

import re

import os

import shutil

if \_\_name\_\_=="\_\_main\_\_":

shutil.copyfile("a.txt","hello2.txt")

shutil.move("a.txt","../")

shutil.move("hello2.txt","aaa.txt")

## 文件与目录的重命名

os模块的函数rename()可以对文件或目录进行重命名

演示文件重命名的操作。如果当前目录存在名为hello.txt的文件，则重命名为hi.txt;如果存在hi.txt的文件则重命名为hello.txt

#-\*-coding:utf-8-\*-

import sys

import os

if \_\_name\_\_=="\_\_main\_\_":

li=os.listdir(".")#判断当前目录

print li

if "hello.txt" in li:

os.rename("hello.txt","hi.txt")

elif "hi.txt" in li:

os.rename("hi.txt","hello.txt")

把后缀名为”html”的文件修改为”htm”后缀的文件

#-\*-coding:utf-8-\*-

import sys

import os

if \_\_name\_\_=="\_\_main\_\_":

files=os.listdir(".")

for filename in files:

#查找文件名中.所在的位置并把它给pos

pos=filename.find(".")

#print pos

#得到.后面的内容

if filename[pos+1:]=="html":

print filename

newname=filename[:pos+1]+"htm"

print newname

os.rename(filename,newname)

#-\*-coding:utf-8-\*-

import sys

import os

if \_\_name\_\_=="\_\_main\_\_":

files=os.listdir(".")

for filename in files:

li=os.path.splitext(filename)

if li[1]==".html":

newname=li[0]+".htm"

os.rename(filename,newname)

## 文件内容的查找和替换

从文件中查找字符串hello，并统计hello出现的次数

#-\*-coding:utf-8-\*-

import sys

import os

import re

if \_\_name\_\_=="\_\_main\_\_":

fl=file("aaa.txt","r")

count=0

for s in fl.readlines():

li=re.findall("hello",s)

if len(li)>0:

count=count+li.count("hello")

print "查找到"+str(count)+"个hello"

fl.close()

把hello.txt中的字符串hello全部替换为hi，并把结果保存在文件hello2.txt中

#-\*-coding:utf-8-\*-

import sys

import os

import re

if \_\_name\_\_=="\_\_main\_\_":

f1=file("aaa.txt","r")

f2=file("bbb.txt","w")

for s in f1.readlines():

f2.write(s.replace("hello","hi"))

f1.close()

f2.close()

## 文件的比较

Python提供了模块difflib用于实现对序列、文件的比较。如果要比较两个文件，列出两个文件的异同，可以使用difflib模块的SequenceMatcher类实现。其中的方法get\_opcodes()可以返回两个序列的比较结果。调用方法get\_opcodes()之前，需要生成1个SequenceMatcher对象。

## 控制台输入

raw\_input控制台输入

#-\*-coding:utf-8-\*-

**import** os

myname **=** raw\_input**(**"please enter you name:"**)**

**print** myname

getpass.getpass方法

#-\*-coding:utf-8-\*-

**import** os

**import** getpass

#简单

myname **=** raw\_input**(**"please enter you name:"**)**

**print** myname

#输入密码安全

pwd **=** getpass**.**getpass**(**'password: '**)**

**print** pwd

# 中文乱码

以下文章引用之：

<http://blog.csdn.net/kiki113/article/details/4062063>

http://www.51testing.com/?uid-524463-action-viewspace-itemid-817888

并适当编排

## 编码普及

### ASCII码

我们知道，在计算机内部，所有的信息最终都表示为一个二进制的字符串。每一个二进制位（bit）有0和1两种状态，因此八个二进制位就可以组合出256种状态，这被称为一个字节（byte）。也就是说，一个字节一共可以用来表示256种不同的状态，每一个状态对应一个符号，就是256个符号，从0000000到11111111。

上个世纪60年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为ASCII码，一直沿用至今。

ASCII码一共规定了128个字符的编码，比如空格“SPACE”是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的1位统一规定为0。

### 非ASCII编码

英语用128个符号编码就够了，但是用来表示其他语言，128个符号是不够的。比如，在法语中，字母上方有注音符号，它就无法用ASCII码表示。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的é的编码为130（二进制10000010）。这样一来，这些欧洲国家使用的编码体系，可以表示最多256个符号。

但是，这里又出现了新的问题。不同的国家有不同的字母，因此，哪怕它们都使用256个符号的编码方式，代表的字母却不一样。比如，130在法语编码中代表了é，在希伯来语编码中却代表了字母Gimel (ג)，在俄语编码中又会代表另一个符号。但是不管怎样，所有这些编码方式中，0—127表示的符号是一样的，不一样的只是128—255的这一段。

至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示256x256=65536个符号。

中文编码的问题需要专文讨论，这篇笔记不涉及。这里只指出，虽然都是用多个字节表示一个符号，但是GB类的汉字编码与后文的Unicode和UTF-8是毫无关系的。

### Unicode

正如上一节所说，世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。因此，要想打开一个文本文件，就必须知道它的编码方式，否则用错误的编码方式解读，就会出现乱码。为什么电子邮件常常出现乱码？就是因为发信人和收信人使用的编码方式不一样。

可以想象，如果有一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是Unicode，就像它的名字都表示的，这是一种所有符号的编码。

Unicode当然是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，U+0639表示阿拉伯字母Ain，U+0041表示英语的大写字母A，U+4E25表示汉字“严”。具体的符号对应表，可以查询unicode.org，或者专门的汉字对应表。

### Unicode的问题

需要注意的是，Unicode只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。

比如，汉字“严”的unicode是十六进制数4E25，转换成二进制数足足有15位（100111000100101），也就是说这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

这里就有两个严重的问题，第一个问题是，如何才能区别unicode和ascii？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？第二个问题是，我们已经知道，英文字母只用一个字节表示就够了，如果unicode统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是0，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

它们造成的结果是：1）出现了unicode的多种存储方式，也就是说有许多种不同的二进制格式，可以用来表示unicode。2）unicode在很长一段时间内无法推广，直到互联网的出现。

### UTF-8

互联网的普及，强烈要求出现一种统一的编码方式。UTF-8就是在互联网上使用最广的一种unicode的实现方式。其他实现方式还包括UTF-16和UTF-32，不过在互联网上基本不用。重复一遍，这里的关系是，UTF-8是Unicode的实现方式之一。

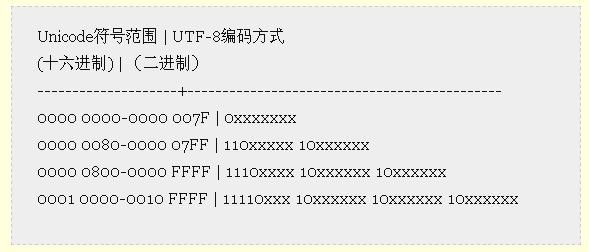
UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的unicode码。因此对于英语字母，UTF-8编码和ASCII码是相同的。

2）对于n字节的符号（n>1），第一个字节的前n位都设为1，第n+1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的unicode码。

下表总结了编码规则，字母x表示可用编码的位。



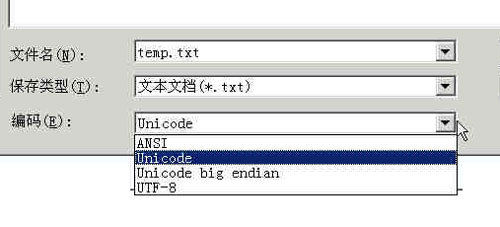
下面，还是以汉字“严”为例，演示如何实现UTF-8编码。

已知“严”的unicode是4E25（100111000100101），根据上表，可以发现4E25处在第三行的范围内（0000 0800-0000 FFFF），因此“严”的UTF-8编码需要三个字节，即格式是“1110xxxx 10xxxxxx 10xxxxxx”。然后，从“严”的最后一个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了，“严”的UTF-8编码是“11100100 10111000 10100101”，转换成十六进制就是E4B8A5。

### Unicode与UTF-8之间的转换

通过上一节的例子，可以看到“严”的Unicode码是4E25，UTF-8编码是E4B8A5，两者是不一样的。它们之间的转换可以通过程序实现。

在Windows平台下，有一个最简单的转化方法，就是使用内置的记事本小程序Notepad.exe。打开文件后，点击“文件”菜单中的“另存为”命令，会跳出一个对话框，在最底部有一个“编码”的下拉条。



里面有四个选项：ANSI，Unicode，Unicode big endian 和 UTF-8。

1）ANSI是默认的编码方式。对于英文文件是ASCII编码，对于简体中文文件是GB2312编码（只针对Windows简体中文版，如果是繁体中文版会采用Big5码）。

2）Unicode编码指的是UCS-2编码方式，即直接用两个字节存入字符的Unicode码。这个选项用的little endian格式。

3）Unicode big endian编码与上一个选项相对应。我在下一节会解释little endian和big endian的涵义。

4）UTF-8编码，也就是上一节谈到的编码方法。

选择完”编码方式“后，点击”保存“按钮，文件的编码方式就立刻转换好了。

### Little endian和Big endian

上一节已经提到，Unicode码可以采用UCS-2格式直接存储。以汉字”严“为例，Unicode码是4E25，需要用两个字节存储，一个字节是4E，另一个字节是25。存储的时候，4E在前，25在后，就是Big endian方式；25在前，4E在后，就是Little endian方式。

这两个古怪的名称来自英国作家斯威夫特的《格列佛游记》。在该书中，小人国里爆发了内战，战争起因是人们争论，吃鸡蛋时究竟是从大头(Big-Endian)敲开还是从小头(Little-Endian)敲开。为了这件事情，前后爆发了六次战争，一个皇帝送了命，另一个皇帝丢了王位。

因此，第一个字节在前，就是”大头方式“（Big endian），第二个字节在前就是”小头方式“（Little endian）。

那么很自然的，就会出现一个问题：计算机怎么知道某一个文件到底采用哪一种方式编码？

Unicode规范中定义，每一个文件的最前面分别加入一个表示编码顺序的字符，这个字符的名字叫做”零宽度非换行空格“（ZERO WIDTH NO-BREAK SPACE），用FEFF表示。这正好是两个字节，而且FF比FE大1。

如果一个文本文件的头两个字节是FE FF，就表示该文件采用大头方式；如果头两个字节是FF FE，就表示该文件采用小头方式。

#### 8. 实例

下面，举一个实例。

打开”记事本“程序Notepad.exe，新建一个文本文件，内容就是一个”严“字，依次采用ANSI，Unicode，Unicode big endian 和 UTF-8编码方式保存。

然后，用文本编辑软件UltraEdit中的”十六进制功能“，观察该文件的内部编码方式。

1）ANSI：文件的编码就是两个字节“D1 CF”，这正是“严”的GB2312编码，这也暗示GB2312是采用大头方式存储的。

2）Unicode：编码是四个字节“FF FE 25 4E”，其中“FF FE”表明是小头方式存储，真正的编码是4E25。

3）Unicode big endian：编码是四个字节“FE FF 4E 25”，其中“FE FF”表明是大头方式存储。

4）UTF-8：编码是六个字节“EF BB BF E4 B8 A5”，前三个字节“EF BB BF”表示这是UTF-8编码，后三个“E4B8A5”就是“严”的具体编码，它的存储顺序与编码顺序是一致的。

## 中文乱码细究

在本文中，以'哈'来解释作示例解释所有的问题，“哈”的各种编码如下：

  1. UNICODE (UTF8-16)，C854；

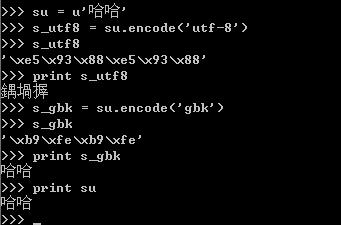
  2． UTF-8，E59388；

  3． GBK，B9FE。

### python中的str和unicode

一直以来，python中的中文编码就是一个极为头大的问题，经常抛出编码转换的异常，python中的str和unicode到底是一个什么东西呢？

在python中提到unicode，一般指的是unicode对象，例如'哈哈'的unicode对象为u'/u54c8/u54c8'而str，是一个字节数组，这个字节数组表示的是对unicode对象编码(可以是utf-8、gbk、cp936、GB2312)后的存储的格式。这里它仅仅是一个字节流，没有其它的含义，如果你想使这个字节流显示的内容有意义，就必须用正确的编码格式，解码显示。例如：



对于unicode对象哈哈进行编码，编码成一个utf-8编码的str－s\_utf8,s\_utf8就是是一个字节数组，存放的就是'/xe5/x93/x88/xe5/x93/x88'，但是这仅仅是一个字节数组，如果你想将它通过print语句输出成哈哈，那你就失望了，为什么呢？

因为print语句它的实现是将要输出的内容传送了操作系统，操作系统会根据系统的编码对输入的字节流进行编码，这就解释了为什么utf-8格式的字符串“哈哈”，输出的是“鍝堝搱”，因为'/xe5/x93/x88/xe5/x93/x88'用GB2312去解释，其显示的出来就是“鍝堝搱”。这里再强调一下，str记录的是字节数组，只是某种编码的存储格式，至于输出到文件或是打印出来是什么格式，完全取决于其解码的编码将它解码成什么样子。

这里再对print进行一点补充说明：当将一个unicode对象传给print时，在内部会将该unicode对象进行一次转换，转换成本地的默认编码（这仅是个人猜测）

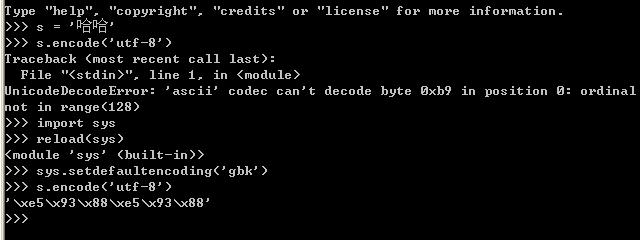
### str和unicode对象的转换

str和unicode对象的转换，通过encode和decode实现，具体使用如下：

decode和encode演示

将GBK'哈哈'转换成unicode，然后再转换成UTF8

## Setdefaultencoding



如上图的演示代码所示：当把s(gbk字符串)直接编码成utf-8的时候，将抛出异常，但是通过调用如下代码：

import sys

reload(sys)

sys.setdefaultencoding('gbk')

后就可以转换成功，为什么呢？在python中str和unicode在编码和解码过程中，如果将一个str直接编码成另一种编码，会先把str解码成unicode，采用的编码为默认编码，一般默认编码是anscii，所以在上面示例代码中第一次转换的时候会出错，当设定当前默认编码为'gbk'后，就不会出错了。至于reload(sys)是因为Python2.5 初始化后会删除 sys.setdefaultencoding 这个方法，我们需要重新载入。

### 操作不同文件的编码格式的文件

建立一个文件test.txt，文件格式用ANSI，内容为:abc中文，用python来读取。

# coding=gbk

print open("Test.txt").read()

结果：abc中文

把文件格式改成UTF-8：

结果：abc涓枃

显然，这里需要解码：

# coding=gbk

import codecs

print open("Test.txt").read().decode("utf-8")

结果：abc中文

上面的test.txt我是用Editplus来编辑的，但当我用Windows自带的记事本编辑并存成UTF-8格式时，

运行时报错：

Traceback (most recent call last):

File "ChineseTest.py", line 3, in

print open("Test.txt").read().decode("utf-8")

UnicodeEncodeError: 'gbk' codec can't encode character u'/ufeff' in position 0: illegal multibyte sequence

原来，某些软件，如notepad，在保存一个以UTF-8编码的文件时，会在文件开始的地方插入三个不可见的字符（0xEF 0xBB 0xBF，即BOM）。

因此我们在读取时需要自己去掉这些字符，python中的codecs module定义了这个常量：

# coding=gbk

import codecs

data = open("Test.txt").read()

if data[:3] == codecs.BOM\_UTF8:

data = data[3:]

print data.decode("utf-8")

结果：abc中文

### 文件的编码格式和编码声明的作用

源文件的编码格式对字符串的声明有什么作用呢？这个问题困扰一直困扰了我好久，现在终于有点眉目了，文件的编码格式决定了在该源文件中声明的字符串的编码格式，例如：

str = '哈哈'

print repr(str)

a.如果文件格式为utf-8，则str的值为：'/xe5/x93/x88/xe5/x93/x88'

b.如果文件格式为gbk，则str的值为：'/xb9/xfe/xb9/xfe'（哈哈的gbk编码）

在第一节已经说过，python中的字符串，只是一个字节数组，所以当把a情况的str输出到gbk编码的控制台时，就将显示为乱码：鍝堝搱；而当把b情况下的str输出utf-8编码的控制台时，也将显示乱码的问题，是什么也没有，也许'/xb9/xfe/xb9/xfe'用utf-8解码显示，就是空白吧。>\_<

说完文件格式，现在来谈谈编码声明的作用吧，每个文件在最上面的地方，都会用# coding=gbk 类似的语句声明一下编码，但是这个声明到底有什么用呢？到止前为止，我觉得它的作用也就是三个：

* 声明源文件中将出现非ascii编码，通常也就是中文；
* 在高级的IDE中，IDE会将你的文件格式保存成你指定编码格式。
* 决定源码中类似于u'哈'这类声明的将‘哈’解码成unicode所用的编码格式，也是一个比较容易让人迷惑的地方，看示例：

#coding:gbk

ss = u'哈哈'

print repr(ss)

print 'ss:%s' % ss

将这个些代码保存成一个utf-8文本，运行，你认为会输出什么呢？大家第一感觉肯定输出的肯定是：

u'/u54c8/u54c8'

ss:哈哈

但是实际上输出是：

u'/u935d/u581d/u6431'

ss:鍝堝搱

为什么会这样，这时候，就是编码声明在作怪了，在运行ss = u'哈哈'的时候，整个过程可以分为以下几步：

1) 获取'哈哈'的编码：由文件编码格式确定，为'/xe5/x93/x88/xe5/x93/x88'（哈哈的utf-8编码形式）

2) 转成unicode编码的时候，在这个转换的过程中，对于'/xe5/x93/x88/xe5/x93/x88'的解码，不是用utf-8解码，而是用声明编码处指定的编码GBK，将'/xe5/x93/x88/xe5/x93/x88'按GBK解码，得到就是''鍝堝搱''，这三个字的unicode编码就是u'/u935d/u581d/u6431'，至止可以解释为什么print repr(ss)输出的是u'/u935d/u581d/u6431'了。

好了，这里有点绕，我们来分析下一个示例：

#-\*- coding:utf-8 -\*-

ss = u'哈哈'

print repr(ss)

print 'ss:%s' % ss

将这个示例这次保存成GBK编码形式，运行结果，竟然是：

UnicodeDecodeError: 'utf8' codec can't decode byte 0xb9 in position 0: unexpected code byte

## 新手编码必读（[gonewithewind](http://www.51testing.com/?uid-524463-action-viewpro-showpro-1)大虾作品）

为什么会报错

“UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not in range(128)”

本文就来研究一下这个问题。字符串在Python内部的表示是unicode编码，因此，在做编码转换时，通常需要以unicode作为中间编码，即先将其他编码的字符串解码（decode）成unicode，再从unicode编码（encode）成另一种编码。

decode的作用是将其他编码的字符串转换成unicode编码，如str1.decode('gb2312')，表示将gb2312编码的字符串str1转换成unicode编码。

encode的作用是将unicode编码转换成其他编码的字符串，如str2.encode('gb2312')，表示将unicode编码的字符串str2转换成gb2312编码。

因此，转码的时候一定要先搞明白，字符串str是什么编码，然后decode成unicode，然后再encode成其他编码，代码中字符串的默认编码与代码文件本身的编码一致。

如：s='中文'

如果是在utf8的文件中，该字符串就是utf8编码，如果是在gb2312的文件中，则其编码为gb2312。这种情况下，要进行编码转换，都需 要先用decode方法将其转换成unicode编码，再使用encode方法将其转换成其他编码。通常，在没有指定特定的编码方式时，都是使用的系统默 认编码创建的代码文件。

如果字符串是这样定义：s=u'中文'，则该字符串的编码就被指定为unicode了，即python的内部编码，而与代码文件本身的编码无关。因此，对于这种情况做编码转换，只需要直接使用encode方法将其转换成指定编码即可。

如果一个字符串已经是unicode了，再进行解码则将出错，因此通常要对其编码方式是否为unicode进行判断：

isinstance(s, unicode)#用来判断是否为unicode，用非unicode编码形式的str来encode会报错。

如何获得系统的默认编码？

#!/usr/bin/env python

#coding=utf-8

import sys

print sys.getdefaultencoding()

该段程序在英文WindowsXP上输出为：ascii

在某些IDE中，字符串的输出总是出现乱码，甚至错误，其实是由于IDE的结果输出控制台自身不能显示字符串的编码，而不是程序本身的问题。

如在UliPad中运行如下代码：

s=u"中文"

print s

会提示：UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not in range(128)。

这是因为UliPad在英文WindowsXP上的控制台信息输出窗口是按照ascii编码输出的（英文系统的默认编码是 ascii），而上面代码中的字符串是Unicode编码的，所以输出时产生了错误。

将最后一句改为：print s.encode('gb2312')则能正确输出“中文”两个字。若最后一句改为：

print s.encode('utf8')

则输出：\xe4\xb8\xad\xe6\x96\x87，这是控制台信息输出窗口按照ascii编码输出utf8编码的字符串的结果。

unicode(str,'gb2312')与str.decode('gb2312')是一样的，都是将gb2312编码的str转为unicode编码。使用str.\_\_class\_\_可以查看str的编码形式

# Web.Py框架概述

## Python下web开发框架的选择

以下内容为网络摘录，来源：飞龙博客

### [Django](http://www.djangoproject.com/)

http://feilong.me/wp-content/uploads/2011/01/hdr_logo.gif

Python框架虽然说是百花齐放，但仍然有那么一家是最大的，它就是Django。要说Django是Python框架里最好的，有人同意也有人坚决反对，但说Django的文档最完善、市场占有率最高、招聘职位最多估计大家都没什么意见。Django为人所称道的地方主要有：

* 完美的文档，Django的成功，我觉得很大一部分原因要归功于Django近乎完美的官方文档（包括Django book）。
* 全套的解决方案，Django象Rails一样，提供全套的解决方案（full-stack framework + batteries included），基本要什么有什么（比如：cache、session、feed、orm、geo、auth），而且全部Django自己造，开发网站应手的工具Django基本都给你做好了，因此开发效率是不用说的，出了问题也算好找，不在你的代码里就在Django的源码里。
* 强大的URL路由配置，Django让你可以设计出非常优雅的URL，在Django里你基本可以跟丑陋的GET参数说拜拜。
* 自助管理后台，admin interface是Django里比较吸引眼球的一项contrib，让你几乎不用写一行代码就拥有一个完整的后台管理界面。

而Django的缺点主要源自Django坚持自己造所有的轮子，整个系统相对封闭，Django最为人诟病的地方有：

* 系统紧耦合，如果你觉得Django内置的某项功能不是很好，想用喜欢的第三方库来代替是很难的，比如下面将要说的ORM、Template。要在Django里用SQLAlchemy或Mako几乎是不可能，即使打了一些补丁用上了也会让你觉得非常非常别扭。
* Django自带的ORM远不如SQLAlchemy强大，除了在Django这一亩三分地，SQLAlchemy是Python世界里事实上的ORM标准，其它框架都支持SQLAlchemy了，唯独Django仍然坚持自己的那一套。Django的开发人员对SQLAlchemy的支持也是有过讨论和尝试的，不过最终还是放弃了，估计是代价太高且跟Django其它的模块很难合到一块。
* Template功能比较弱，不能插入Python代码，要写复杂一点的逻辑需要另外用Python实现Tag或Filter。关于模板这一点，一直以来争论比较多，最近有两篇关于Python模板的比较有意思的文章可供参考：

1. <http://pydanny.blogspot.com/2010/12/stupid-template-languages.html>（需翻墙）
2. <http://techspot.zzzeek.org/2010/12/04/in-response-to-stupid-template-languages/>

* URL配置虽然强大，但全部要手写，这一点跟Rails的Convention over configuration的理念完全相左，高手和初识Django的人配出来的URL会有很大差异。
* 让人纠结的auth模块，Django的auth跟其它模块结合紧密，功能也挺强的，就是做的有点过了，用户的数据库schema都给你定好了，这样问题就来了，比如很多网站要求email地址唯一，可schema里这个字段的值不是唯一的，纠结是必须的了。
* Python文件做配置文件，而不是更常见的ini、xml或yaml等形式。这本身不是什么问题，可是因为理论上来说settings的值是能够动态的改变的（虽然大家不会这么干），但这不是最佳实践的体现。

总的来说，Django大包大揽，用它来快速开发一些Web运用是很不错的。如果你顺着Django的设计哲学来，你会觉得Django很好用，越用越爽；相反，你如果不能融入或接受Django的设计哲学，你用Django一定会很痛苦，趁早放弃的好。所以说在有些人眼里Django无异于仙丹，但对有一些人来说它又是毒药且剧毒。

Django案例有[disqus.com](http://disqus.com/)、[bitbucket.org](http://bitbucket.org/)、[海报网](http://www.haibao.cn/)等。

### [Pylons](http://www.pylonshq.com/) & [TurboGears](http://turbogears.org/) & [repoze.bfg](http://bfg.repoze.org/)



除了Django另一个大头就是Pylons了，因为TurboGears2.x是基于Pylons来做的，而repoze.bfg也已经并入Pylons project里这个大的项目里，后面不再单独讨论TurboGears和repoze.bfg了。

Pylons和Django的设计理念完全不同，Pylons本身只有两千行左右的Python代码，不过它还附带有一些几乎就是Pylons御用的第三方模块。Pylons只提供一个架子和可选方案，你可以根据自己的喜好自由的选择Template、ORM、form、auth等组件，系统高度可定制。我们常说Python是一个胶水语言(glue language)，那么我们完全可以说Pylons就是一个用胶水语言设计的胶水框架:)

选择Pylons多是选择了它的自由，选择了自由的同时也预示着你选择了噩梦：

* 学习噩梦，Pylons依赖于许多第三方库，它们并不是Pylons造，你学Pylons的同时还得学这些库怎么使用，关键有些时候你都不知道你要学什么。Pylons的学习曲线相对比Django要高的多，而之前Pylons的官方文档也一直是人批评的对象，好在后来出了[The Definitive Guide to Pylons](http://pylonsbook.com/en/1.1/)这本书，这一局面有所改观。因为这个原因，Pylons一度被誉为只适合高手使用的Python框架。
* 调试噩梦，因为牵涉到的模块多，一旦有错误发生就比较难定位问题处在哪里。可能是你写的程序的错、也可能是Pylons出错了、再或是SQLAlchemy出错了、搞不好是formencode有bug，反正很凌乱了。这个只有用的很熟了才能解决这个问题。
* 升级噩梦，安装Pylons大大小小共要安装近20个Python模块，各有各自的版本号，要升级Pylons的版本，哪个模块出了不兼容的问题都有可能，升级基本上很难很难。至今reddit的Pylons还停留在古董的0.9.6上，SQLAlchemy也还是0.5.3的版本，应该跟这条有关系。所以大家玩Pylons一定要结合virtualenv来玩，给自己留条后路，不然会死得很惨。

Pylons和repoze.bfg的融合可能会催生下一个能挑战Django地位的框架。

Pylons的案例有[reddit.com](http://www.reddit.com/)、[dropbox.com](http://www.dropbox.com/)、[quora.com](http://www.quora.com/)等。

### [Tornado](http://www.tornadoweb.org/)& [web.py](http://webpy.org/)



Tornado即是一个web server（对此本文不作详述），同时又是一个类web.py的micro-framework，作为框架Tornado的思想主要来源于web.py，大家在web.py的网站首页也可以看到Tornado的大佬[Bret Taylor](http://bret.appspot.com/)的这么一段话（他这里说的FriendFeed用的框架跟Tornado可以看作是一个东西）：

“[web.py inspired the] web framework we use at FriendFeed [and] the webapp framework that ships with App Engine…”

因为有这层关系，后面不再单独讨论Tornado。

web.py的设计理念力求精简（Keep it simple and powerful），总共就没多少行代码，也不像Pylons那样依赖大量的第三方模块，而是只提供的一个框架所必须的一些东西，如：URL路由、Template、数据库访问，其它的就交给用户自己去做好了。

一个框架精简的好处在于你可以聚焦在业务逻辑上，而不用太多的去关心框架本身或受框架的干扰，同时缺点也很明显，许多事情你得自己操刀上。  
我个人比较偏好这种精简的框架，因为你很容易通过阅读源码弄明白整个框架的工作机制，如果框架那一块不是很合意的话，我完全可以Monkey patch一下按自己的要求来。

早期的reddit是用web.py写的，Tornado的案例有[friendfeed.com](http://friendfeed.com/)、[bit.ly](http://bit.ly/)、[quora.com](http://www.quora.com/)和我的开源站点[poweredsites.org](http://poweredsites.org/)等。

### [Bottle](http://bottle.paws.de/) & [Flask](http://flask.pocoo.org/)



Bottle和Flask作为新生一代Python框架的代表，挺有意思的是都采用了decorator的方式配置URL路由，如：

**from** bottle **import** route, run

@route('/:name')

**def** index(name='World'):

**return** '&lt;b&gt;Hello %s!&lt;/b&gt;' % name

run(host='localhost', port=8080)

Bottle、Flask跟web.py一样，都非常精简，Bottle甚至所有的代码都在那一个两千来行的.py文件里。另外Flask和Pylons一样，可以跟Jinja2、SQLAlchemy之类结合的很好。

不过目前不管是Bottle还是Flask成功案例都还很少。

### [Quixote](http://www.quixote.ca/)

之所以要特别说一下Quixote，是因为国内的最大的用Python开发的网站“[豆瓣网](http://www.douban.com/)”是用Quixote开发的。我只简单翻了一下源代码，没有做过研究，不发表评论，有经验的来补充下。我只是在想，如果豆瓣网交到现在来开发，应该会有更多的选择。

### 最后关于框架选择的误区

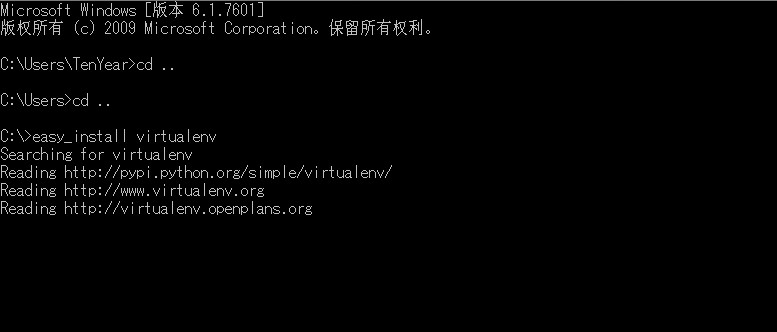
在框架的选择问题上，许多人很容易就陷入了下面两个误区中而不自知：

1. 哪个框架最好 － 世上没有最好的框架，只有最适合你自己、最适合你的团队的框架。编程语言选择也是一个道理，你的团队Python最熟就用Python好了，如果最熟悉的是Ruby那就用Ruby好了，编程语言、框架都只是工具，能多、快、好、省的干完活就是好东西，管TMD是日本鬼子还是美帝造呢！
2. 过分关注性能 － 其实大部分人是没必要太关心框架的性能的，因为你开发的网站根本就是个小站，能上1万的IP的网站已经不多了，上10万的更是很少很少。在没有一定的访问量前谈性能其实是没有多大意义的，因为你的CPU和内存一直就闲着呢。而且语言和框架一般也不会是性能瓶颈，性能问题最常出现在数据库访问和文件读写上。PHP的Zend Framework是出了名的慢，但是Zend Framework一样有大站，如：digg.com；常被人说有性能问题的Ruby和Rails，不是照样可以开发出twitter吗？再者现在的硬件、带宽成本其实是很低的，特别有了云计算平台后，人力成本才是最贵的，没有上万的IP根本就不用太在意性能问题，流量上去了花点钱买点服务器空间好了，简单快速的解决性能问题。

注：前面有网友质疑我“Quora是用Pylons开发的”这样的说法不客观，特说明一下，这里所说的某个网站A是用B开发的，只是指A主要或部分是由B开发的，大家就不要再去纠结A还用C了。

## 解决SAE第三方包的安装方案

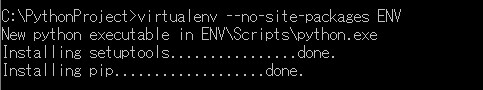
1. 安装virtualenv



1. 安装pip



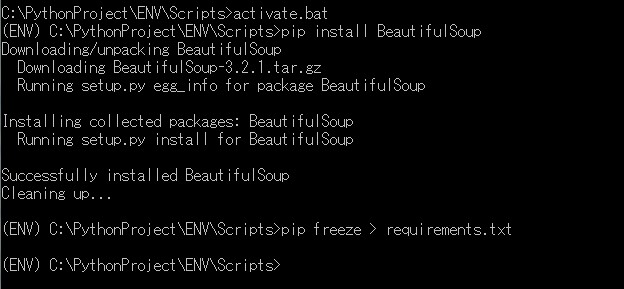
1. 创建一个Python的虚拟环境，不依赖于本机的Python原有环境



1. 这时会在当前目录生成一个ENV目录，进入这个目录中

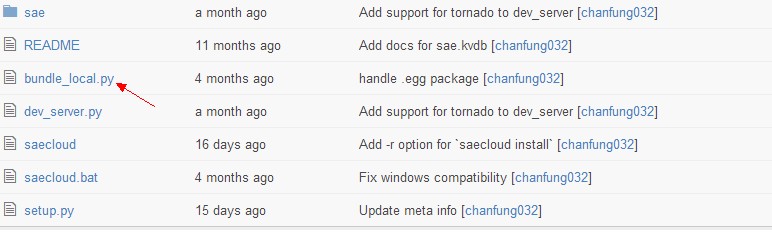


1. 下载所需要打包的第三方包，并且冻结



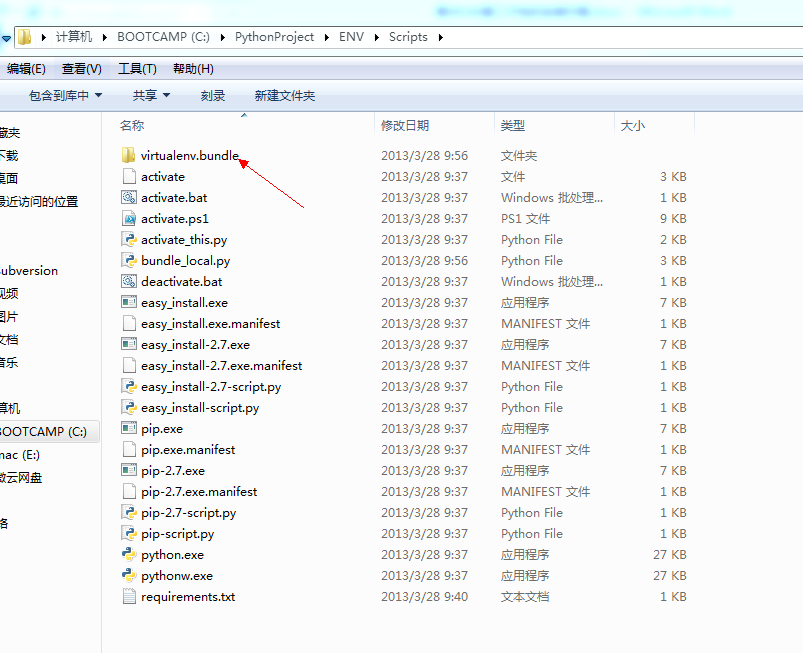
1. 需要去Github下载服务器打包工具。

https://github.com/SAEPython/saepythondevguide/tree/master/dev\_server



1. 打包

python bundle\_local.py -r requirements.txt



将virutalenv.bundle目录到应用的目录下。修改index.wsgi文件，在导入其它模块之前，将virtualenv.bundle目录或者 virtualenv.bundle.zip添加到module的搜索路径中，示例代码如下：

import os

import sys

app\_root = os.path.dirname(\_\_file\_\_)

# 两者取其一

sys.path.insert(0,os.path.join(app\_root,'virtualenv.bundle'))

到此，所有的依赖包已经导出并加入到应用的目录里了。

## web.py安装

首先下载Web.Py的安装包：

http://webpy.org/static/web.py-0.37.tar.gz

或者从GITHUB下载最新的安装包也可以：

https://github.com/webpy/webpy/tarball/master

使用命令行安装

python setup.py install

如果是Linux下面的话，需要使用Sudo

sudo python setup.py install

## web.py下常用框架简介

* jinja2，python中一个很流行的模板框架，用了它，写模板就是如此的享受。
* sqlalchemy，python最强大的orm，没有之一，掌握了它，数据库？so easy！
* formalchemy，配套sqlalchemy的表单框架，可以根据sqlalchemy中数据表的定义生成html表单
* kendo ui，这是一个jquery框架，提供了很多插件，里面最让我欣赏的，就是mvvm的开发模式和datasource的ajax，让本来已经很少的jquery代码变得更少了。

## 最简单的HelloWorld!

#-\*- coding:utf-8 -\*-

**import** web

#定义url，将地址映射到对应的类

urls **=** **(**

"/"**,** "index"**,**

**)**

app **=** web**.**application**(**urls**,** globals**())**

#定义index类

**class** **index:**

#get请求

**def** GET**(**self**):**

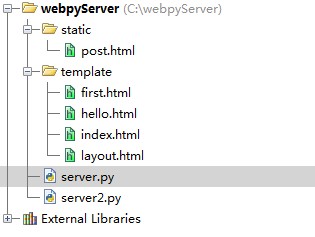
**return** "Hello World"

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

app**.**run**()**

## Web.py基本使用模板使用

项目结构



server.py

#-\*-coding:utf-8-\*-

import web

urls = (

'/', 'index'

)

render = web.template.render('template/')

class index:

def GET(self):

#return render.hello('楚广明')

return render.first('world','name')

if \_\_name\_\_ == "\_\_main\_\_":

app = web.application(urls, globals())

app.run()

first.html

$def with (name1,name2)

<html><head>

<title>my si

</title>

</head><body>

Hello $name1 , hello $name2!

<table>

$for c in ["a", "b", "c", "d"]:

<tr class="$loop.parity">

<td>$loop.index</td>

<td>$c</td>

</tr>

</table>

</body></html>

## 进阶模板使用

### server2.py

#-\*-coding:utf-8-\*-

import web

from web import form as form

urls = (

'/', 'index',

'/add','add'

)

render = web.template.render('template/')

class index:

def GET(self):

return render.post()

class add:

def POST(self):

print web.input()['title1']

print web.data()

raise web.seeother('/')

if \_\_name\_\_ == "\_\_main\_\_":

app = web.application(urls, globals())

app.run()

### post.html

<html><head>

<title>post test

</title>

</head><body>

<form method="post" action="/add">

<input type="text" name="title" />

<input type="text" name="title1" />

<input type="text" name="title2" />

<input type="text" name="title3" />

<input type="submit" value="Add" />

</form>

</body></html>

## URL控制

### 问题:

如何为整个网站设计一个URL控制方案 / 调度模式

### 解决:

web.py的URL控制模式是简单的、强大的、灵活的。在每个应用的最顶部，你通常会看到整个URL调度模式被定义在元组中:

urls = (

"/tasks/?", "signin",

"/tasks/list", "listing",

"/tasks/post", "post",

"/tasks/chgpass", "chgpass",

"/tasks/act", "actions",

"/tasks/logout", "logout",

"/tasks/signup", "signup"

)

这些元组的格式是: URL路径, 处理类 这组定义有多少可以定义多少。如果你并不知道URL路径和处理类之间的关系。

### 路径匹配

你可以利用强大的正则表达式去设计更灵活的URL路径。比如 /(test1|test2) 可以捕捉 /test1 或 /test2。要理解这里的关键，匹配是依据URL路径的。比如下面的URL:

http://localhost/myapp/greetings/hello?name=Joe

你可以捕捉URL的参数，然后用在处理类中:

/users/list/(.+), "list\_users"

在 list/后面的这块会被捕捉，然后作为参数被用在GET或POST:

class list\_users:

def GET(self, name):

return "Listing info about user: {0}".format(name)

## 一个完整的小例子

#-\*-coding:utf-8-\*-

import web

urls = (

"/", "hello",

"/chu888/?","chu888",

"/parmtest/(.+)","listmy"

)

app = web.application(urls, globals())

class hello:

def GET(self):

return 'Hello, world!'

class chu888:

def GET(self):

return "你访问的是chu888"

class listmy:

def GET(self,name):

return "Listing info about user: {0}".format(name)

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

## web.seeother 和 web.redirect转向

### 问题

在处理完用户输入后（比方说处理完一个表单），如何跳转到其他页面？

### 解法

class SomePage:

def POST(self):

# Do some application logic here, and then:

raise web.seeother('/someotherpage')

POST方法接收到一个post并完成处理之后，它将给浏览器发送一个303消息和新网址。接下来，浏览器就会对这个新网址发出GET请求，从而完成跳转。

注意：web.seeother和web.redirect不支持0.3以下版本。

### 区别

用web.redirect方法似乎也能做同样的事情，但通常来说，这并太友好。因为web.redirect发送的是301消息－这是永久重定向。因为大多数Web浏览器会缓存新的重定向，所以当我们再次执行该操作时，会自动直接访问重定向的新网址。很多时候，这不是我们所想要的结果。所以在提交表单时，尽量使用seeother。但是在下面要提到的这种场合，用redirect却是最恰当的：我们已经更改了网站的网址结构，但是仍想让用户书签/收藏夹中的旧网址不失效。

(注：要了解seeother和redirect的区别，最好是看一下http协议中不同消息码的含义。)

## 包含应用

### 实现

在blog.py中:

import web

urls = (

"", "reblog",

"/(.\*)", "blog"

)

class reblog:

def GET(self): raise web.seeother('/')

class blog:

def GET(self, path):

return "blog " + path

app\_blog = web.application(urls, locals())

当前的主应用code.py:

import web

import blog

urls = (

"/blog", blog.app\_blog,

"/(.\*)", "index"

)

class index:

def GET(self, path):

return "hello " + path

app = web.application(urls, locals())

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

## 使用XML

### 问题

如何在web.py中提供XML访问？

如果需要为第三方应用收发数据，那么提供xml访问是很有必要的。

### 解法

根据要访问的xml文件(如response.html)创建一个XML模板。如果XML中有变量，就使用相应的模板标签进行替换。下面是一个例子：

$def with (code)

<?xml version="1.0"?>

<RequestNotification-Response>

<Status>$code</Status>

</RequestNotification-Response>

为了提供这个XML，需要创建一个单独的web.py程序(如response.py)，它要包含下面的代码。注意：要用"web.header('Content-Type', 'text/xml')"来告知客户端－－正在发送的是一个XML文件。

import web

render = web.template.render('template/', cache=False)

urls = (

'/(.\*)', 'index'

)

app = web.application(urls, globals())

class index:

def GET(self, code):

web.header('Content-Type', 'text/xml')

return render.index(code)

web.webapi.internalerror = web.debugerror

if \_\_name\_\_ == '\_\_main\_\_': app.run()

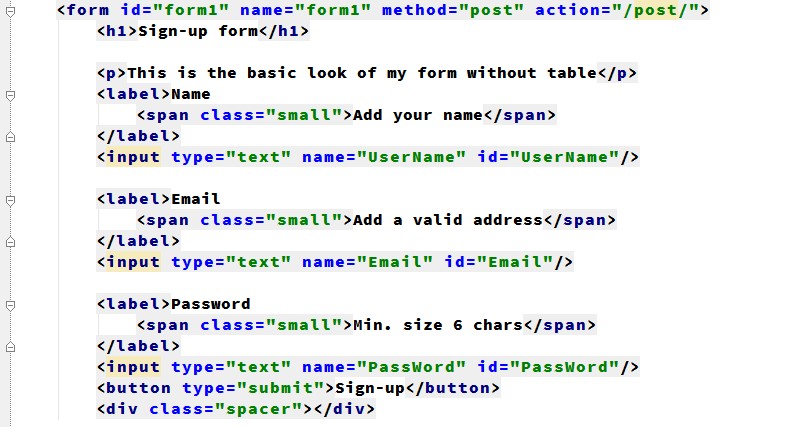
## 获取POST数据

class RequestHandler():

def POST():

data = web.data() # 通过这个方法可以取到数据

### login.html模板



### 主程序

#-\*-coding:utf-8-\*-

import web

urls = (

"/", "index",

"/login/?","login",

"/post/?","post"

)

render = web.template.render('template/')

app = web.application(urls, globals())

class index:

def GET(self):

return "Hello web.py!"

class login:

def GET(self):

return render.login()

class post:

def POST(self):

return web.input()['UserName']

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

## 获取客户端信息

### 问题

如何在代码中得到客户端信息？比如：来源页面(referring page)或是客户端浏览器类型

### 解法

使用web.ctx即可。首先讲一点架构的东西：web.ctx基于threadeddict类，又被叫做ThreadDict。这个类创建了一个类似字典(dictionary-like)的对象，对象中的值都是与线程id相对应的。这样做很妙,因为很多用户同时访问系统时，这个字典对象能做到仅为某一特定的HTTP请求提供数据(因为没有数据共享，所以对象是线程安全的)

web.ctx保存每个HTTP请求的特定信息，比如客户端环境变量。假设，我们想知道正在访问某页面的用户是从哪个网页跳转而来的：

### 例子

class example:

def GET(self):

referer = web.ctx.env.get('HTTP\_REFERER', 'http://google.com')

raise web.seeother(referer)

上述代码用web.ctx.env获取HTTP\_REFERER的值。如果HTTP＿REFERER不存在，就会将google.com做为默认值。接下来，用户就会被重定向回到之前的来源页面。

web.ctx另一个特性，是它可以被loadhook赋值。例如：当一个请求被处理时，会话(Session)就会被设置并保存在web.ctx中。由于web.ctx是线程安全的，所以我们可以象使用普通的python对象一样，来操作会话(Session)。

### 'ctx'中的数据成员

Request

* environ 又被写做. env – 包含标准WSGI环境变量的字典。
* home – 应用的http根路径(译注：可以理解为应用的起始网址，协议＋站点域名＋应用所在路径)例：http://example.org/admin
* homedomain – 应用所在站点(可以理解为协议＋域名) http://example.org
* homepath – 当前应用所在的路径，例如： /admin
* host – 主机名（域名）＋用户请求的端口（如果没有的话，就是默认的80端口），例如： example.org,example.org:8080
* ip – 用户的IP地址，例如： xxx.xxx.xxx.xxx
* method – 所用的HTTP方法，例如： GET
* path – 用户请求路径，它是基于当前应用的相对路径。在子应用中，匹配外部应用的那部分网址将被去掉。例如：主应用在code.py中，而子应用在admin.py中。在code.py中, 我们将/admin关联到admin.app。 在admin.py中, 将/stories关联到stories类。在 stories中, web.ctx.path就是/stories, 而非/admin/stories。形如： /articles/845
* protocol – 所用协议，例如： https
* query – 跟在'？'字符后面的查询字符串。如果不存在查询参数，它就是一个空字符串。例如： ?fourlegs=good&twolegs=bad
* fullpath 可以视为 path + query – 包含查询参数的请求路径，但不包括'homepath'。例如：/articles/845?fourlegs=good&twolegs=bad

Response

* status – HTTP状态码（默认是'200 OK') 401 Unauthorized 未经授权
* headers – 包含HTTP头信息(headers)的二元组列表。
* output – 包含响应实体的字符串。

## 文件上传

### 简单的文件上传

#!/usr/bin/env python

# coding: utf-8

import web

class mytest:

def GET(self):

return "<b><h1>我扩展测试一下</h1><b>"

class Upload:

def GET(self):

return """<html><head></head><body>

<form method="POST" enctype="multipart/form-data" action="">

<input type="file" name="myfile" />

<br/>

<input type="submit" />

</form>

</body></html>"""

def POST(self):

x = web.input(myfile={})

web.debug(x['myfile'].filename) # 这里是文件名

web.debug(x['myfile'].value) # 这里是文件内容

web.debug(x['myfile'].file.read()) # 或者使用一个文件对象

print x['myfile'].filename

print x['myfile'].value

print x['myfile'].file.read()

raise web.seeother('/todo/upload')

### 基于新浪云的上传

#!/usr/bin/env python

# coding: utf-8

import web

import uuid

import random

import time

import sae.storage

from config import settings

from datetime import datetime

render = settings.render

class SaveUpload:

def GET(self):

# 初始化一个Storage客户端。

s = sae.storage.Client()

imagelist=s.list('pythondb')

return render.SaveUpload(imagelist)

# def GET(self):

# web.header("Content-Type","text/html; charset=utf-8")

# return """<html><head><title>文件上传并且保存demo</title></head><body>

# <form method="POST" enctype="multipart/form-data" action="">

# <input type="file" name="myfile" />

# <br/>

# <input type="submit" />

# </form>

# </body></html>"""

def POST(self):

x = web.input(myfile={})

if 'myfile' in x:

#读取所有文件内容

imagevalue=x.myfile.file.read()

# 初始化一个Storage客户端。

s = sae.storage.Client()

# 设置object的属性

ob = sae.storage.Object(imagevalue)

#保存文件

saveuploadfile= str(time.time()) + str(random.random())+x.myfile.filename

s.put('pythondb',saveuploadfile, ob)

raise web.seeother('/todo/saveupload')

$def with(imagelist)

<!DOCTYPE html>

<html>

<head>

<title></title>

<style type="text/css">

.imagediv

{

width: 200px;

height: 200px;

border: solid 1px;

}

</style>

</head>

<body>

<form method="POST" enctype="multipart/form-data" action="">

<input type="file" name="myfile" />

<br/>

<input type="submit" />

</form>

$for i in imagelist:

<div ><img class="imagediv" src="http://chinapython-pythondb.stor.sinaapp.com/$i['name']"/></div>

</body>

</html>

# Web.py使用Session

## 问题

如何在web.py中使用session

## 解法

**注意**！！！：session并不能在调试模式(Debug mode)下正常工作，这是因为session与调试模试下的重调用相冲突(有点类似firefox下著名的Firebug插件，使用Firebug插件分析网页时，会在火狐浏览器之外单独对该网页发起请求，所以相当于同时访问该网页两次)，下一节中我们会给出在调试模式下使用session的解决办法。

web.session模块提供session支持。下面是一个简单的例子－－统计有多少人正在使用session(session计数器)：

import web

web.config.debug = False

urls = (

"/count", "count",

"/reset", "reset"

)

app = web.application(urls, locals())

session = web.session.Session(app, web.session.DiskStore('sessions'), initializer={'count': 0})

class count:

def GET(self):

session.count += 1

return str(session.count)

class reset:

def GET(self):

session.kill()

return ""

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

web.py在处理请求之前，就加载session对象及其数据；在请求处理完之后，会检查session数据是否被改动。如果被改动，就交由session对象保存。

上例中的initializer参数决定了session初始化的值，它是个可选参数。如果用数据库代替磁盘文件来存储session信息，只要用DBStore代替DiskStore即可。使用DBStore需要建立一个表，结构如下：

create table sessions (

session\_id char(128) UNIQUE NOT NULL,

atime timestamp NOT NULL default current\_timestamp,

data text

);

DBStore被创建要传入两个参数：db对象和session的表名。

db = web.database(dbn='postgres', db='mydatabase', user='myname', pw='')

store = web.session.DBStore(db, 'sessions')

session = web.session.Session(app, store, initializer={'count': 0})

｀web.config｀中的sessions\_parameters保存着session的相关设置，sessions\_parameters本身是一个字典，可以对其修改。默认设置如下：

web.config.session\_parameters['cookie\_name'] = 'webpy\_session\_id'

web.config.session\_parameters['cookie\_domain'] = None

web.config.session\_parameters['timeout'] = 86400, #24 \* 60 \* 60, # 24 hours in seconds

web.config.session\_parameters['ignore\_expiry'] = True

web.config.session\_parameters['ignore\_change\_ip'] = True

web.config.session\_parameters['secret\_key'] = 'fLjUfxqXtfNoIldA0A0J'

web.config.session\_parameters['expired\_message'] = 'Session expired'

* cookie\_name - 保存session id的Cookie的名称
* cookie\_domain - 保存session id的Cookie的domain信息
* timeout - session的有效时间 ，以秒为单位
* ignore\_expiry - 如果为True，session就永不过期
* ignore\_change\_ip - 如果为true，就表明只有在访问该session的IP与创建该session的IP完全一致时，session才被允许访问。
* secret\_key - 密码种子，为session加密提供一个字符串种子
* expired\_message - session过期时显示的提示信息。

## 在template下使用Session

### 问题:

我想在模板中使用session（比如：读取并显示session.username）

### 解决:

在应用程序中的代码:

render = web.template.render('templates', globals={'context': session})

在模板中的代码:

<span>You are logged in as <b>$context.username</b></span>

你可以真正的使用任何符合语法的python变量名，比如上面用的context。我更喜欢在应用中直接使用'session'。

server.py

#-\*-coding:utf-8-\*-

import web

web.config.debug = False

urls = (

"/count", "count",

"/reset", "reset",

"/","index"

)

app = web.application(urls, locals())

db = web.database(dbn='mysql', db='todo', user='root', pw='chu123')

store = web.session.DBStore(db, 'sessions')

session = web.session.Session(app, store, initializer={'count': 0})

render = web.template.render('template', globals={'context': session})

class index:

def GET(self):

return render.index()

class count:

def GET(self):

session.count += 1

return str(session.count)

class reset:

def GET(self):

session.kill()

return ""

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

index.html

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<h1>我是楚广明</h1>

<h1><span>You are logged in as <b>$context.count</b></span></h1>

</body>

</html>

## 如何操作Cookies

### 问题

如何设置和获取用户的Cookie?

### 解法

对web.py而言，设置/获取Cookie非常方便。

### 设置Cookies

#### 概述

setcookie(name, value, expires="", domain=None, secure=False):

* name (string) - Cookie的名称，由浏览器保存并发送至服务器。
* value (string) -Cookie的值，与Cookie的名称相对应。
* expires (int) - Cookie的过期时间，这是个可选参数，它决定cookie有效时间是多久。以秒为单位。它必须是一个整数，而绝不能是字符串。
* domain (string) - Cookie的有效域－在该域内cookie才是有效的。一般情况下，要在某站点内可用，该参数值该写做站点的域（比如.webpy.org），而不是站主的主机名（比如wiki.webpy.org）
* secure (bool)- 如果为True，要求该Cookie只能通过HTTPS传输。.

#### 示例

用web.setcookie() 设置cookie,如下:

class CookieSet:

def GET(self):

i = web.input(age='25')

web.setcookie('age', i.age, 3600)

return "Age set in your cookie"

用 GET方式调用上面的类将设置一个名为age,默认值是25的cookie(实际上，默认值25是在web.input中赋予i.age的，从而间接赋予 cookie，而不是在setcookie函式中直接赋予cookie的)。这个cookie将在一小时后(即3600秒)过期。

web.setcookie()的第三个参数－"expires"是一个可选参数，它用来设定cookie过期的时间。如果是负数，cookie将立刻过期。如果是正数，就表示cookie的有效时间是多久，以秒为单位。如果该参数为空，cookie就永不过期。

### 获得Cookies

#### 概述

获取Cookie的值有很多方法，它们的区别就在于找不到cookie时如何处理。

##### 方法1（如果找不到cookie，就返回None）：

web.cookies().get(cookieName)

#cookieName is the name of the cookie submitted by the browser

##### 方法2（如果找不到cookie，就抛出AttributeError异常）：

foo = web.cookies()

foo.cookieName

##### 方法3（如果找不到cookie，可以设置默认值来避免抛出异常）：

foo = web.cookies(cookieName=defaultValue)

foo.cookieName # return the value (which could be default)

#cookieName is the name of the cookie submitted by the browser

#### 示例：

用web.cookies() 访问cookie. 如果已经用web.setcookie()设置了Cookie, 就可以象下面这样获得Cookie:

class CookieGet:

def GET(self):

c = web.cookies(age="25")

return "Your age is: " + c.age

这个例子为cookie设置了默认值。这么做的原因是在访问时，若cookie不存在，web.cookies()就会抛出异常，如果事先设置了默认值就不会出现这种情况。

如果要确认cookie值是否存在，可以这样做：

class CookieGet:

def GET(self):

try:

return "Your age is: " + web.cookies().age

except:

# Do whatever handling you need to, etc. here.

return "Cookie does not exist."

或

class CookieGet:

def GET(self):

age=web.cookies().get(age)

if age:

return "Your age is: %s" % age

else:

return "Cookie does not exist."

## 布局模板

### 问题

如何让站点每个页面共享一个整站范围的模板？（在某些框架中，称为模板继承，比如ASP.NET中的母版页）

### 方法

我们可以用 base 属性来实现:

render = web.template.render('templates/', base='layout')

现在如果你调用render.foo()方法，将会加载templates/foo.html 模板，并且它将会被templates/layout.html模板包裹。

"layout.html" 是一个简单模板格式文件，它包含了一个模板变量，如下:

$def with (content)

<html>

<head>

<title>Foo</title>

</head>

<body>

$:content

</body>

</html>

在某些情况，如果不想使用基本模板，只需要创建一个没有base属性的reander对象，如下：

render\_plain = web.template.render('templates/')

程序

#-\*-coding:utf-8-\*-

import web

web.config.debug = True

urls = (

"/","index"

)

app = web.application(urls, locals())

render = web.template.render('template/', base='layout')

class index:

def GET(self):

return render.index('楚广明6666')

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

layout.html

$def with (content)

<html>

<head>

<title>Foo</title>

</head>

<body>

$:content

</body>

</html>

index.html

$def with (name)

<h1>我是$name</h1>

### Tip: 在布局文件（layout.html）中定义的页面标题变量

##### templates/index.html

$var title: This is title.

<h3>Hello, world</h3>

##### templates/layout.html

$def with (content)

<html>

<head>

<title>$content.title</title>

</head>

<body>

$:content

</body>

</html>

### Tip: 在其他模板中引用css文件，如下:

#### templates/login.html

$var cssfiles: static/login.css static/login2.css

hello, world.

#### templates/layout.html

$def with (content)

<html>

<head>

<title>$content.title</title>

$if content.cssfiles:

$for f in content.cssfiles.split():

<link rel="stylesheet" href="$f" type="text/css" media="screen" charset="utf-8"/>

</head>

<body>

$:content

</body>

</html>

输入的HTML代码如下:

<link rel="stylesheet" href="static/login.css" type="text/css" media="screen" charset="utf-8"/>

<link rel="stylesheet" href="static/login2.css" type="text/css" media="screen" charset="utf-8"/>

# Web.py支持的语法模板

## 使用Templetor模板语法

### Introduction

web.py 的模板语言叫做 Templetor，它能负责将 python 的强大功能传递给模板系统。 在模板中没有重新设计语法，它是类 python 的。 如果你会 python，你可以顺手拈来。

这是一个模板示例:

$def with (name)

Hello $name!

第一行表示模板定义了一个变量 name。 第二行中的 $name 将会用 name 的值来替换。

### 使用模板系统

通用渲染模板的方法：

render = web.template.render('templates')

return render.hello('world')

render 方法从模板根目录查找模板文件，render.hello(..)表示渲染 hello.html 模板。实际上，系统会在根目录去查找叫 hello的所有文件，直到找到匹配的。(事实上他只支持 .html 和 .xml 两种)

完整程序：

#-\*-coding:utf-8-\*-

import web

web.config.debug = True

urls = (

"/","index"

)

app = web.application(urls, locals())

render = web.template.render('template', globals)

class index:

def GET(self):

return render.index('楚广明6666')

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

模板：

$def with (name)

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<h1>我是$name</h1>

</body>

</html>

除了上面的使用方式，你也可以直接用文件的方式来处理模板 frender：

hello = web.template.frender('templates/hello.html')

render hello('world')

直接使用字符串方式：

template = "$def with (name)\nHello $name"

hello = web.template.Template(template)

return hello('world')

### 表达式用法

特殊字符 $ 被用于特殊的 python 表达式。表达式能够被用于一些确定的组合当中 () 和 {}:

Look, a $string.

Hark, an ${arbitrary + expression}.

Gawk, a $dictionary[key].function('argument').

Cool, a $(limit)ing.

### 赋值

有时你可能需要定义一个新变量或给一些变量重新赋值，如下：

$ bug = get\_bug(id)

<h1>$bug.title</h1>

<div>

$bug.description

<div>

**注意 $在赋值变量名称之前要有一个空格，这有区别于常规的赋值用法。**

**完整程序：**

#-\*-coding:utf-8-\*-

import web

web.config.debug = True

urls = (

"/","index"

)

app = web.application(urls, locals())

render = web.template.render('template', globals)

class index:

def GET(self):

return render.index('楚广明6666')

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

$def with (name)

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<h1>我是$name</h1>

$def get\_bug(id='1'):

Hello $name!

$ bug = get\_bug(id)

<h1>$bug</h1>

</body>

</html>

### 过滤

模板默认会使用 web.websafe 过滤 html 内容(encodeing 处理)。

>>> render.hello("1 < 2")

"Hello 1 &lt; 2"

不需要过滤可以在 $ 之后 使用 :。示例：该 Html 内容不会被转义

$:form.render()

### 新起一行用法

在行末添加 \ 代表显示层该内容不会被真实处理成一行。

If you put a backslash \

at the end of a line \

(like these) \

then there will be no newline.

### 转义 $

使用 $$ 可以在输出的时候显示字符 $.

Can you lend me $$50?

### 注释

$# 是注释指示符。任何以 $# 开始的某行内容都被当做注释。

$# this is a comment

Hello $name.title()! $# display the name in title case

### 控制结构

**模板系统支持 for, while, if, elif 和 else。像 python 一样，这里是需要缩进的。**

$for i in range(10):

I like $i

$for i in range(10): I like $i

$ a= [1,2,3,4,5]

$while a:

hello $a.pop()

$ times=100

$ max=200

$if times > max:

Stop! In the name of love.

$else:

Keep on, you can do it.

for 循环内的成员变量只在循环内发生可用：

loop.index: the iteration of the loop (1-indexed)

loop.index0: the iteration of the loop (0-indexed)

loop.first: True if first iteration

loop.last: True if last iteration

loop.odd: True if an odd iteration

loop.even: True if an even iteration

loop.parity: "odd" or "even" depending on which is true

loop.parent: the loop above this in nested loops

有时候，他们使用起来很方便：

<table>

$for c in ["a", "b", "c", "d"]:

<tr class="$loop.parity">

<td>$loop.index</td>

<td>$c</td>

</tr>

</table>

### 使用 def

可以使用 $def 定义一个新的模板函数，支持使用参数。

$def say\_hello(name='world'):

Hello $name!

$say\_hello('web.py')

$say\_hello()

其他示例：

$def tr(values):

<tr>

$for v in values:

<td>$v</td>

</tr>

$def table(rows):

<table>

$for row in rows:

$:row

</table>

$ data = [['a', 'b', 'c'], [1, 2, 3], [2, 4, 6], [3, 6, 9] ]

$:table([tr(d) for d in data])

#### 代码

可以在 code 块书写任何 python 代码：

$code:

x = "you can write any python code here"

y = x.title()

z = len(x + y)

def limit(s, width=10):

"""limits a string to the given width"""

if len(s) >= width:

return s[:width] + "..."

else:

return s

And we are back to template.

The variables defined in the code block can be used here.

For example, $limit(x)

### 使用 var

var 块可以用来定义模板结果的额外属性：

$def with (title, body)

$var title: $title

$var content\_type: text/html

<div id="body">

$body

</div>

以上模板内容的输出结果如下：

>>> out = render.page('hello', 'hello world')

>>> out.title

u'hello'

>>> out.content\_type

u'text/html'

>>> str(out)

'\n\n<div>\nhello world\n</div>\n'

### 内置 和 全局

像 python 的任何函数一样，模板系统同样可以使用内置以及局部参数。很多内置的公共方法像range，min，max等，以及布尔值 True 和 False，在模板中都是可用的。部分内置和全局对象也可以使用在模板中。

全局对象可以使用参数方式传给模板，使用 web.template.render：

import web

import markdown

globals = {'markdown': markdown.markdown}

render = web.template.render('templates', globals=globals)

内置方法是否可以在模板中也是可以被控制的：

# 禁用所有内置方法

render = web.template.render('templates', builtins={})

这个比较难理解，我可以给出一个例子：

模板很简单index.html

$:title

code.py

#-\*- coding:utf-8 -\*-

**import** web

**import** sys

#如果页面有修改可以即时体现

reload**(**sys**)**

sys**.**setdefaultencoding**(**'utf-8'**)**

#定义url，将地址映射到对应的类

urls **=** **(**

"/"**,** "index"**,**

**)**

app **=** web**.**application**(**urls**,** globals**())**

**def** render**(**params**={},** partial**=False):**

global\_vars **=** dict**(**params**.**items**())**

**if** partial**:**

#print "true"

**return** web**.**template**.**render**(**'templates/'**,** globals**=**global\_vars**)**

**else:**

#print "false"

**return** web**.**template**.**render**(**'templates/'**,** base**=**'layout'**,** globals**=**global\_vars**)**

#定义index类

**class** **index:**

#get请求

**def** GET**(**self**):**

**return** render**({**'title'**:**'这是一个测试'**},True).**index**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

app**.**run**()**

## 使用Jinja2模板语法

### 上手的一个小例子

#-\*- coding:utf-8 -\*-

**import** web

# 引入web.py对调用jinja2的模块

**from** web**.**contrib**.**template **import** render\_jinja

urls **=** **(**

"/"**,** "index"**,**

**)**

app **=** web**.**application**(**urls**,** globals**())**

render **=** render\_jinja**(**

'templates'**,** # 模板存放的目录名称

encoding**=**'utf-8'**,** # 模板使用的编码

**)**

**class** **index:**

**def** GET**(**self**):**

**return** render**.**index**(**word**=**"Hello World"**)** # 使用模板目录下的index.html模板

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

app**.**run**()**

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<h1>{{ word }}</h1>

</body>

</html>

# Web.py中使用数据库

## Mysql-Python在Ubuntu下的安装

sudo apt-get install python-setuptools

sudo apt-get install python-dev

sudo apt-get install  libmysqld-dev

sudo apt-get install libmysqlclient-dev

sudo easy\_install -U distribute

sudo easy\_install mysql-python

## 使用web.py自带的DB封装库

首先需要大家用easy\_install安装mysql的驱动，也可以自行去下载软件。有一定要注意就是64位的驱动与32位的是不兼容的，这一点要注意了。

web.py是一个非常精巧的web框架，不过其自带的db模块也是非常精简而高效。

### 一个简单的数据库访问测试

#-\*-coding:utf-8-\*-

import sys

import web

#定义数据库对象db

db = web.database(dbn='mysql', db='todolist', user='root', pw='chu123')

#定义要操作的数据库名称

tb = 'todo'

def get\_by\_id(id):

s = db.select(tb, where='id=$id', vars=locals())

if not s:

return False

return s[0]

if \_\_name\_\_=="\_\_main\_\_":

resultValue=get\_by\_id(4)

print resultValue

**如果语句里面使用了$符号，就要添加vars=locals()这一句，不然可能会报错。里面的$id是传递过来的id变量。**

### 一个复杂一点的例子

#-\*-coding:utf-8-\*-

import sys

import web

import time

#定义数据库对象db

db = web.database(dbn='mysql', db='todolist', user='root', pw='chu123')

#定义要操作的数据库名称

tb = 'todo'

def getTableInfo(id=1):

result = db.query('select \* from todo where id>$id', vars=locals())

for s in result:

print s.id, s.title

if \_\_name\_\_=="\_\_main\_\_":

#简单查询

result = db.query('select \* from todo where id>$id', vars={'id':1})

for s in result:

print s.id, s.title

#插入,第一参数是表名todo

db.insert('todo', title='Michael',finished=0,post\_date=time.strftime('%Y-%m-%d',time.localtime(time.time())))

getTableInfo(5)

#更新

db.update('todo', where='id=$id', vars={'id':5}, title='楚广明')

getTableInfo(5)

#删除

db.delete('todo', where='id=$id', vars={'id':6})

getTableInfo(5)

### 更加复杂的例子

#-\*-coding:utf-8-\*-

import sys

import web

import time

#定义数据库对象db

db = web.database(dbn='mysql', db='todolist', user='root', pw='chu123')

#定义要操作的数据库名称

tb = 'todo'

if \_\_name\_\_=="\_\_main\_\_":

# 查询表

entries = db.select('todo')

# where 条件

myvar = dict(title="楚广明")

results = db.select('todo', myvar, where="title = $title")

for s in results:

print s.id

results = db.select('todo', where="id>100")

for s in results:

print s.id

# 查询具体列

results = db.select('todo', what="id,title")

# order by

results = db.select('todo', order="post\_date DESC")

# group

results = db.select('todo', group="title")

# limit

results = db.select('todo', limit=10)

#update

db.update('todo', where="id = 10", title = "foo")

#delete

db.delete('todo', where="id=10")

#复杂

# count

results = db.query("SELECT COUNT(\*) AS total\_users FROM todo")

print results[0].total\_users

# join

#results = db.query("SELECT \* FROM entries JOIN users WHERE entries.author\_id = users.id")

# 防止SQL注入可以这么干

#results = db.query("SELECT \* FROM users WHERE id=$id", vars={'id':10})

'''多数据库操作

db1 = web.database(dbn='mysql', db='dbname1', user='foo')

db2 = web.database(dbn='mysql', db='dbname2', user='foo')

print db1.select('foo', where='id=1')

print db2.select('bar', where='id=5')

'''

'''事务

t = db.transaction()

try:

db.insert('person', name='foo')

db.insert('person', name='bar')

except:

t.rollback()

raise

else:

t.commit()

# Python 2.5+ 可以用with

from \_\_future\_\_ import with\_statement

with db.transaction():

db.insert('person', name='foo')

db.insert('person', name='bar')

'''

# 使用Sqlalchemy

## 简介

web开发中一般的主要版块就是视图、模板、数据库操作，前面已经讲了最基础的视图跟模板，这一章主要是介绍下数据库操作，如第一章前言所说，我们的数据库操作是使用sqlalchemy来实现的。

对象关系映射器（Object Relational Mappers，ORM）在过去数年吸引了不少人的目光。主要原因是 ORM 经常会在 Web 应用程序框架中被提起，因为它是快速开发（Rapid Development）栈中的关键组件。Django 和 Ruby on Rails 等 Web 框架采用了设计一个独立栈的方法，将自主开发的 ORM 紧密集成到该框架中。而其他框架，如 Pylons、Turbogears 和 Grok，则采用更加基于组件的架构结合可交换的第三方组件。两种方法都有各自的优势：紧密集成允许非常连贯的体验（如果问题映射到框架），而基于组件的架构则允许最大的设计灵活性。但是，本文的主题并不是 Web 框架；而是 SQLAlchemy。

SQLAlchemy 在构建在 WSGI 规范上的下一代 Python Web 框架中得到了广泛应用，它是由 Mike Bayer 和他的核心开发人员团队开发的一个单独的项目。使用 ORM 等独立 SQLAlchemy 的一个优势就是它允许开发人员首先考虑数据模型，并能决定稍后可视化数据的方式（采用命令行工具、Web 框架还是 GUI 框架）。这与先决定使用 Web 框架或 GUI 框架，然后再决定如何在框架允许的范围内使用数据模型的开发方法极为不同。

不同于很多web开发，python开发web的数据库很多是通过建立数据表类，绑定数据库连接，然后自动创建数据表，这种设计很大程度上降低了在开发阶段修改数据库结构的麻烦，而且由于表是对应一个个类创建而成，可以很清晰的在任何时候了解到该app的数据库是如何设计的，这对维护的意义也是非常大的。

**什么是 WSGI？**

WSGI 是下一代 Python Web 框架、应用程序和服务器应该遵循的规范。WSGI 中一个有趣的方面是创建 Python 中间件，并在使用 Python 或任何语言创建的 Web 应用程序中使用。请参阅 参考资料，获取关于 WSGI 和 WSGI 社区（Pypefitters）的大量链接。

SQLAlchemy 的一个目标是提供能兼容众多数据库（如 SQLite、MySQL、Postgres、Oracle、MS-SQL、SQLServer 和 Firebird）的企业级持久性模型。SQLAlchemy 正处于积极开发阶段，当前最新的 API 将围绕版本 0.5 设计。请参阅参考资料部分，获取官方 API 文档、教程和 SQLAlchemy 书籍的链接。

SQLAlchemy 取得成功的一个证明就是围绕它已建立了丰富的社区。针对 SQLAlchemy 的扩展和插件包括：declarative、Migrate、Elixir、SQLSoup、django-sqlalchemy、DBSprockets、FormAlchemy 和 z3c.sqlalchemy。在本文中，我们将学习一篇关于新 0.5 API 的教程，探究一些第三方库，以及如何在 Pylons 中使用它们。

**谁是 Mike Bayer？**

Michael Bayer 是居住在纽约的一名软件承包商，他拥有十余年处理各类关系数据库的经验。他曾使用 C、Java™ 和 Perl 编写了许多自主研发的数据库抽象层，并在 Major League Baseball 与大量多服务器 Oracle 系统打了多年交道，借助这些经验，他成功编写了 “终极工具包” SQLAlchemy，用于生成 SQL 和处理数据库。其目标是贡献一个世界级、独树一帜的面向 Python 的工具包，以帮助 Python 成为一个广泛普及的编程平台。

例子一

# -\*- coding:utf-8 -\*-

**from** sqlalchemy **import** **\***

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** scoped\_session**,** sessionmaker

db\_config **=** **{**

'host'**:** 'localhost'**,**

'user'**:** 'root'**,**

'passwd'**:** 'sdss'**,**

'db'**:**'test'**,**

'charset'**:**'utf8'

**}**

engine **=** create\_engine**(**'mysql://%s:%s@%s/%s?charset=%s'**%(**db\_config**[**'user'**],**

db\_config**[**'passwd'**],**

db\_config**[**'host'**],**

db\_config**[**'db'**],**

db\_config**[**'charset'**]),** echo**=True)**

metadata **=** MetaData**()**

users\_table **=** Table**(**'users222'**,** metadata**,**

Column**(**'id'**,** Integer**,** primary\_key**=True),**

Column**(**'name'**,** String**(**50**)),**

Column**(**'fullname'**,** String**(**50**)),**

Column**(**'password'**,** String**(**100**))**

**)**

metadata**.**create\_all**(**engine**)**

例子二

# -\*- coding:utf-8 -\*-

**from** sqlalchemy **import** **\***

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** scoped\_session**,** sessionmaker

#定义数据库的账号、端口、密码、数据库名，使用的连接模块，这里用的是mysqldb

engine **=** create\_engine**(**

'mysql+mysqldb://root:dddd123@localhost:3306/todo?charset=utf8'**,**

echo**=True**#是否输出数据库操作过程，很方便调试

**)**

Base **=** declarative\_base**()**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** "user2222"

id **=** Column**(**Integer**,** primary\_key**=True)**

name **=** Column**(**String**(**20**),** unique**=True)**

email **=** Column**(**String**(**32**),** unique**=True)**

password **=** Column**(**String**(**32**))**

superuser **=** Column**(**Boolean**,** default**=False)**

metadata **=** Base**.**metadata

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

metadata**.**create\_all**(**engine**)**#运行python models.py就会自动创建定义的所有表

以上就是User表的定义，可以在User类里非常清晰的看出这个数据表是如何设计的。接下来先在mysql创建一个叫"tech"的数据库，接着只要在命令行运行python models.py就可以看到数据库中出现了User表了，用数据库操作工具看看是否与定义的内容一样吧:)

## 初始化Sqlite数据库

**from** sqlalchemy **import** create\_engine

**from** sqlalchemy **import** Column**,** Integer**,** String

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

engine **=** create\_engine**(**'sqlite:///mydatabase.db'**,** echo**=True)**

Base **=** declarative\_base**()**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** 'users'

id **=** Column**(**Integer**,** primary\_key**=True)**

name **=** Column**(**String**)**

fullname **=** Column**(**String**)**

password **=** Column**(**String**)**

**def** \_\_init\_\_**(**self**,** name**,** fullname**,** password**):**

self**.**name **=** name

self**.**fullname **=** fullname

self**.**password **=** password

**def** \_\_repr\_\_**(**self**):**

**return** "<User('%s','%s', '%s')>" **%** **(**self**.**name**,** self**.**fullname**,** self**.**password**)**

users\_table **=** User**.**\_\_table\_\_

metadata **=** Base**.**metadata

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

metadata**.**create\_all**(**engine**)**

## 插入数据库

# -\*- coding:utf-8 -\*-

**from** sqlalchemy **import** **\***

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** scoped\_session**,** sessionmaker

# 定义数据库的账号、端口、密码、数据库名，使用的连接模块，这里用的是mysqldb

db\_config **=** **{**

'host'**:** 'localhost'**,**

'user'**:** 'root'**,**

'passwd'**:** 'sdss'**,**

'db'**:**'todo'**,**

'charset'**:**'utf8'

**}**

engine **=** create\_engine**(**'mysql://%s:%s@%s/%s?charset=%s'**%(**db\_config**[**'user'**],**

db\_config**[**'passwd'**],**

db\_config**[**'host'**],**

db\_config**[**'db'**],**

db\_config**[**'charset'**]),** echo**=True)**

# 定义一个函数，用来获取sqlalchemy的session

**def** bindSQL**():**

**return** scoped\_session**(**sessionmaker**(**bind**=**engine**))**

Base **=** declarative\_base**()**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** "user22"

id **=** Column**(**Integer**,** primary\_key**=True)**

name **=** Column**(**String**(**20**),** unique**=True)**

email **=** Column**(**String**(**32**),** unique**=True)**

password **=** Column**(**String**(**32**))**

superuser **=** Column**(**Boolean**,** default**=False)**

**def** \_\_init\_\_**(**self**,**name**=**''**,**email**=**''**,**password**=**''**,**superuser**=True):**

self**.**name **=** name

self**.**email **=** email

self**.**password **=** password

self**.**superuser **=** superuser

metadata **=** Base**.**metadata

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

metadata**.**create\_all**(**engine**)** # 运行python models.py就会自动创建定义的所有表

my\_vp **=** User**(**'楚广3明'**,** 'ss3林@sss.com'**,** '22月333'**,False)**

session**=**bindSQL**()**

session**.**add**(**my\_vp**)**

session**.**commit**()**

除了将user表创建出来，我们并没有进行其他的操作，而往往我们的web开发，总需要在网站中设计一个超级管理员，这个超级管理员不是注册出来的，而是在数据库创建初期就有的，那么，我们可以在models.py中创建user表，并同时向user表插入超级管理员的信息，而在插入超级管理员或者以后的修改密码、新增用户的时候，我们的密码都是需要加密的，那么这时就可以添加一个事件触发，当插入设置password字段时，自动加密，好，需求有了，接下来就是行动了，我们把原来的models.py修改成以下代码:

# -\*- coding:utf-8 -\*-

**from** sqlalchemy **import** **\***

**from** sqlalchemy **import** event

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** scoped\_session**,** sessionmaker

**import** hashlib

# 这里定义一个password加密混淆

password\_prefix **=** "Ad%cvcsadefr^!deaf"

# 定义数据库的账号、端口、密码、数据库名，使用的连接模块，

# 这里用的是mysqldb

engine **=** create\_engine**(**

'mysql+mysqldb://root:dddd123@localhost:3306/todo?charset=utf8'**,**

echo**=True** # 是否输出数据库操作过程，很方便调试

**)**

# 定义一个函数，用来获取sqlalchemy的session

**def** bindSQL**():**

**return** scoped\_session**(**sessionmaker**(**bind**=**engine**))**

Base **=** declarative\_base**()**

# 定义数据表使用InnoDB

Base**.**\_\_table\_args\_\_ **=** **{**'mysql\_engine'**:** 'InnoDB'**}**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** "user5555"

id **=** Column**(**Integer**,** primary\_key**=True)**

name **=** Column**(**String**(**20**),** unique**=True)**

email **=** Column**(**String**(**32**),** unique**=True)**

password **=** Column**(**String**(**32**))**

superuser **=** Column**(**Boolean**,** default**=False)**

metadata **=** Base**.**metadata

# 定义一个回调函数用于响应触发事件

**def** setPassword**(**target**,** value**,** oldvalue**,** initiator**):**

# 如果新设置的值与原有的值相等，那么说明用户并没有修改密码，返回原先的值

**if** value **==** oldvalue**:**

**return** oldvalue

# 如果新值与旧值不同，说明密码发生改变，进行加密，加密方法可以根据自己需求改变

**return** hashlib**.**md5**(**"%s%s" **%** **(**password\_prefix**,** value**)).**hexdigest**()**

# 设置事件监听，event.listen(表单或表单字段, 触发事件, 回调函数, 是否改变插入值)

event**.**listen**(**User**.**password**,** "set"**,** setPassword**,** retval**=True)**

# 为了避免重复插入数据，定义一个get\_or\_create函数

**def** get\_or\_create**(**session**,** model**,** **\*\***kwargs**):**

**if** "defaults" **in** kwargs**:**

defaults **=** kwargs**[**"defaults"**]**

**del** kwargs**[**"defaults"**]**

**else:**

defaults **=** **{}**

instance **=** session**.**query**(**model**).**filter\_by**(\*\***kwargs**).**first**()**

**if** instance**:**

**return** instance**,** **False**

**else:**

kwargs**.**update**(**defaults**)**

instance **=** model**(\*\***kwargs**)**

session**.**add**(**instance**)**

session**.**flush**()**

session**.**refresh**(**instance**)**

**return** instance**,** **True**

# 定义初始化函数

**def** initModel**():**

metadata**.**create\_all**(**engine**)** # 创建数据库

db **=** bindSQL**()** # 获取sqlalchemy的session

# 创建超级管理员，这里为了避免多次运行initModel

# 而发生重复插入的情况，使用了get\_or\_create方法

obj**,** created **=** get\_or\_create**(**

db**,**

User**,**

name**=**"administrator"**,**

defaults**={**

"email"**:** "332535694@qq.com"**,**

"password"**:** "administrator"**,**

"superuser"**:** **True**

**}**

**)**

db**.**commit**()** # 记得commit喔，不然数据最后还是没插入

db**.**remove**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

initModel**()**

## 在视图中使用sqlalchemy

### 首先创建数据库models.py

**from** sqlalchemy **import** create\_engine

**from** sqlalchemy **import** Column**,** Integer**,** String

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

engine **=** create\_engine**(**'sqlite:///mydatabase.db'**,** echo**=True)**

Base **=** declarative\_base**()**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** 'users'

id **=** Column**(**Integer**,** primary\_key**=True)**

name **=** Column**(**String**)**

fullname **=** Column**(**String**)**

password **=** Column**(**String**)**

**def** \_\_init\_\_**(**self**,** name**,** fullname**,** password**):**

self**.**name **=** name

self**.**fullname **=** fullname

self**.**password **=** password

**def** \_\_repr\_\_**(**self**):**

**return** "<User('%s','%s', '%s')>" **%** **(**self**.**name**,** self**.**fullname**,** self**.**password**)**

users\_table **=** User**.**\_\_table\_\_

metadata **=** Base**.**metadata

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

metadata**.**create\_all**(**engine**)**

### 使用app.py

**import** string

**import** random

**import** web

**from** sqlalchemy**.**orm **import** scoped\_session**,** sessionmaker

**from** models **import** **\***

urls **=** **(**

"/"**,** "add"**,**

"/view"**,** "view"

**)**

**def** load\_sqla**(**handler**):**

web**.**ctx**.**orm **=** scoped\_session**(**sessionmaker**(**bind**=**engine**))**

**try:**

**return** handler**()**

**except** web**.**HTTPError**:**

web**.**ctx**.**orm**.**commit**()**

**raise**

**except:**

web**.**ctx**.**orm**.**rollback**()**

**raise**

**finally:**

web**.**ctx**.**orm**.**commit**()**

app **=** web**.**application**(**urls**,** globals**())**

app**.**add\_processor**(**load\_sqla**)**

**class** **add:**

**def** GET**(**self**):**

web**.**header**(**'Content-type'**,** 'text/html'**)**

fname **=** ""**.**join**(**random**.**choice**(**string**.**letters**)** **for** i **in** range**(**4**))**

lname **=** ""**.**join**(**random**.**choice**(**string**.**letters**)** **for** i **in** range**(**7**))**

u **=** User**(**name**=**fname**,** fullname**=**fname **+** ' ' **+** lname**,** password**=**542**)**

web**.**ctx**.**orm**.**add**(**u**)**

**return** "added:" **+** web**.**websafe**(**str**(**u**))** \

**+** "<br/>" \

**+** '<a href="/view">view all</a>'

**class** **view:**

**def** GET**(**self**):**

web**.**header**(**'Content-type'**,** 'text/plain'**)**

**return** "\n"**.**join**(**map**(**str**,** web**.**ctx**.**orm**.**query**(**User**).**all**()))**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

app**.**run**()**

# Json的使用

首先Python内置的json包是支持json的解析的，我们平时如果不涉及复杂的业务，这个内置的包功能完全够用。先上一个例子

#!/usr/bin/python

#codeing:utf-8

import json

#Function:Analyze json script

#Json is a script can descript data structure as xml,

#for detail, please refer to "http://json.org/json-zh.html".

#Note:

#1.Also, if you write json script from python,

#you should use dump instead of load. pleaser refer to "help(json)".

#json file:

#The file content of temp.json is:

#{

# "name":"00\_sample\_case1",

# "description":"an example."

#}

#f = file("temp.json");

#s = json.load(f)

#print s

#f.close

#json string:

s = json.loads('{"name":"test", "type":{"name":"seq", "parameter":["1", "2"]}}')

print s

print s.keys()

print s["name"]

print s["type"]["name"]

print s["type"]["parameter"][1]

# Socket编程

# Urllib模块

## urlopen的基本使用

urllib模块提供接口用来打开URL,让我们可以访问www和ftp上的数据并且可以像访问本地文件一样操作他们。

urllib.urlopen(url[, data[, proxies]]) :

url: 表示远程数据的路径

data: 以post方式提交到url的数据

proxies:用于设置代理

urllib.urlopen(url[,date[,poxies]])这个方法打开一个URL并返回远程url的类文件对象，我们可以像操作本地文件一样操作url类的文件对象获取数据，其中参数url是远程数据的地址（URL），date 是以post方式提交到服务器的数据，poxies是设置代理端口的；

urlopen返回的类文件对象有以下方法：

1 read(),readline(),readlines()这些方法跟操作文件一样；

2 fileno() 以整数返回文件描述符；

3 close() 关闭链接；

4 getcode() 返回HTTP响应码，例如：成功会返回200，未找到文件的返回404；

5 geturl() 用于返回请求的URL；

#coding:utf-8

**import** urllib

**import** sys

google**=**urllib**.**urlopen**(**"http://www.baidu.com"**)**

#获取服务器的表头信息

**print** "this is a header:\n%s"**%**google**.**info**()**

**print** "------------------------------------------"

#返回整数状态码

**print** "this is a status:\n%s"**%**google**.**getcode**()**

**print** "-------------------------------------------"

#返回url

**print** "this is a url:\n%s"**%**google**.**geturl**()**

**print** "-------------------------------------------"

#返回文件描述信息

**print** google**.**read**().**decode**(**'utf-8'**)**

#print google.read()

#获取系统默认编码

#print sys.getdefaultencoding()

## urlretrieve方法的使用

urllib.urlretrieve(url[, filename[, reporthook[, data]]])：

filename指定保存到本地的路径（若未指定该，urllib生成一个临时文件保存数据）

reporthook回调函数，当连接上服务器、以及相应的数据块传输完毕的时候会触发该回调

data指post到服务器的数据

urllib.urlretrieve(url[,filename[,reporthook[,date]]]) 此函数将远程的数据下载到本地，其中参数url是URL字符串，filename指定数据保存到本地的路径；reporthook(block,size,total)是一个回调函数当链接上服务器，以及相应的数据传送完毕时会触发该函数（即每下载一块(block)就调用一次回调函数），我们可以用该函数来显示当前数据下载的进度，也可以用来限速，date是指post到服务器的数据；该方法返回一个包含两个元素的元组(filename, headers)，filename表示保存到本地的路径，header表示服务器的响应头。

**from** \_\_future\_\_ **import** unicode\_literals

**import** urllib

**import** re

**def** callback\_f**(**downloaded\_size**,** block\_size**,** romote\_total\_size**):**

per **=** 100.0 **\*** downloaded\_size **\*** block\_size **/** romote\_total\_size

**if** per **>** 100**:**

per **=** 100

**print**"%.2f%%"**%** per

**def** getHtml**(**url**):**

page**=**urllib**.**urlopen**(**url**)**

html**=**page**.**read**()**

page**.**close**()**

**return** html

**def** getImg**(**html**):**

imgre**=**re**.**compile**(**r"""<img\s.\*?\s?src\s\*=\s\*['|"]?([^\s'"]+).\*?>"""**,**re**.**I**)**

imglist**=**imgre**.**findall**(**html**)**

i**=**0

**for** imgurl **in** imglist**:**

**if** **not** imgurl**.**find**(**"http://"**):**

urllib**.**urlretrieve**(**imgurl**,**"img/%d.jpg"**%(**i**),**callback\_f**)**

i**+=**1

**print** imgurl

ht**=**getHtml**(**r"http://www.cnbeta.com/articles/234303.htm"**)**

getImg**(**ht**)**

## url编码

1 urllib.quote(str[，safe]) 对字符串进行编码，而参数safe指定不需要编码的字符；

2 urllib.unquote(str) 对字符串进行解码；

3 urllib.quote\_plus(str[,safe]) 和quote类似,只是把参数str中的空格转化成“+” 而quote是把

空格转化成“%20”

4 urllib.unquote\_plus(str) 对字符串进行解码；

5 urllib.urlencode(query[,dosep]) 参数query可以是两个元素的元组也可以是一个字,将其转成url；

6 urllib.url2pathname(path) 将url路径转化成本地路径；

7 urllib.pathname2url 将本地路径转化成url路径；

#coding:utf-8

**import** urllib

date**=**"whoiam=i am student+i like python"

#对字符串编码

date1**=**urllib**.**quote**(**date**)**

**print** date1

#对字符串编码

**print** urllib**.**unquote**(**date1**)**

**print** urllib**.**unquote\_plus**(**date1**)**

#将参数试点转成url

date5**=**urllib**.**urlencode**({**"name"**:**"python"**,**"love"**:**"python"**})**

**print** date5

## 使用httplib抓取

httplib.HTTPConnection ( host [ , port [ ,strict [ , timeout ]]] )

host表示服务器主机

port为端口号，默认值为80

strict的 默认值为false， 表示在无法解析服务器返回的状态行时(status line) （比较典型的状态行如： HTTP/1.0 200 OK ），是否抛BadStatusLine 异常

可选参数timeout 表示超时时间。

HTTPConnection提供的方法：

- HTTPConnection.request ( method , url [ ,body [ , headers ]] )

调用request 方法会向服务器发送一次请求

method 表示请求的方法，常用有方法有get 和post ；

url 表示请求的资源的url ；

body 表示提交到服务器的数据，必须是字符串（如果method是”post”，则可以把body 理解为html 表单中的数据）；

headers 表示请求的http 头。

- HTTPConnection.getresponse ()

获取Http 响应。返回的对象是HTTPResponse 的实例，关于HTTPResponse 在下面会讲解。

- HTTPConnection.connect ()

连接到Http 服务器。

- HTTPConnection.close ()

关闭与服务器的连接。

- HTTPConnection.set\_debuglevel ( level )

设置高度的级别。参数level 的默认值为0 ，表示不输出任何调试信息。

httplib.HTTPResponse

-HTTPResponse表示服务器对客户端请求的响应。往往通过调用HTTPConnection.getresponse()来创建，它有如下方法和属性：

-HTTPResponse.read([amt])

获取响应的消息体。如果请求的是一个普通的网页，那么该方法返回的是页面的html。可选参数amt表示从响应流中读取指定字节的数据。

-HTTPResponse.getheader(name[, default])

获取响应头。Name表示头域(header field)名，可选参数default在头域名不存在的情况下作为默认值返回。

-HTTPResponse.getheaders()

以列表的形式返回所有的头信息。

-HTTPResponse.msg

获取所有的响应头信息。

-HTTPResponse.version

获取服务器所使用的http协议版本。11表示http/1.1；10表示http/1.0。

-HTTPResponse.status

获取响应的状态码。如：200表示请求成功。

-HTTPResponse.reason

返回服务器处理请求的结果说明。一般为”OK”

#!/usr/bin/python

# -\*- coding:utf-8 -\*-

**def** use\_httplib**():**

**import** httplib

conn **=** httplib**.**HTTPConnection**(**"www.baidu.com"**)**

i\_headers **=** **{**"User-Agent"**:**

"Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1) Gecko/20090624 Firefox/3.5"**,**

"Accept"**:** "text/plain"**}**

conn**.**request**(**"GET"**,** "/"**,** headers**=**i\_headers**)**

r1 **=** conn**.**getresponse**()**

**print**"version:"**,** r1**.**version

**print** "------------------------"

**print**"reason:"**,** r1**.**reason

**print** "------------------------"

**print**"status:"**,** r1**.**status

**print** "------------------------"

**print**"msg:"**,** r1**.**msg

**print** "------------------------"

**print**"headers:"**,** r1**.**getheaders**()**

**print** "------------------------"

data **=** r1**.**read**()**

**print** data**.**decode**(**"utf-8"**)**

**print** "------------------------"

**print** "data length:%s"**%**len**(**data**)**

conn**.**close**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

use\_httplib**()**

# Urllib2的使用

## 最简单的爬虫

网络爬虫是一个自动提取网页的程序，它为搜索引擎从万维网上下载网页，是搜索引擎的重要组成。python的urllib\urllib2等模块很容易实现这一功能，下面的例子实现的是对baidu首页的下载。具体代码如下：

**import** urllib2

page**=**urllib2**.**urlopen**(**"http://www.baidu.com"**)**

**print** page**.**read**().**decode**(**'utf-8'**)**

## 提交表单数据

### 用GET方法提交数据

提交表单的GET方法是把表单数据编码至URL。在给出请示的页面后，加上问号，接着是表单的元素。

如在百度中搜索“马伊琍”得到url为：

<http://www.baidu.com/s?wd=%E9%A9%AC%E4%BC%8A%E7%90%8D&pn=100&rn=20&ie=utf-8&usm=4&rsv_page=1>

其中？后面为表单元素。wd=%E9%A9%AC%E4%BC%8A%E7%90%8D表示搜索的词是“马伊琍”，pn表示从第100条信息所在页开始显示（感觉是这样，我试了几次，当写100时，从其所在页显示，但如果写10，就是从第1页显示），rn=20表示每页显示20条，ie=utf-8表示编码格式，usm=4没明白是什么意思，换了1、2、3试了下，没发现什么变化，rsv\_page=1表示第几页。如果要下载以上页面比较简单的方法是直接用上面的网址进行提取。如代码：

#coding:utf-8

**import** urllib2

keyword**=**urllib2**.**quote**(**'马伊琍'**)**

page**=**urllib2**.**urlopen**(**"http://www.baidu.com/s?wd="**+**keyword**+**"&pn=100&rn=20&ie=utf-8&usm=4&rsv\_page=1"**)**

**print** page**.**read**()**

# Html的解析

在python中能够进行html和xhtml的库有很多，如HTMLParser、sgmllib、htmllib、BeautifulSoup、mxTidy、uTidylib等，这里介绍一下HTMLParser、BeautifulSoup等模块。

## BeautifulSoup进行解析

一段最简单的程序

#coding:utf-8

**import** urllib2

**from** BeautifulSoup **import** BeautifulSoup

page **=** urllib2**.**urlopen**(**'http://www.baidu.com'**);**

soup **=** BeautifulSoup**(**page**)**

#打印页面的编码

**print** soup**.**originalEncoding

#打印规范化的html

**print** soup**.**prettify**().**decode**(**'utf-8'**)**

抓取网页图片并下载

#coding:utf-8

**import** urllib2

**import** urllib

**import** uuid

**from** BeautifulSoup **import** BeautifulSoup

**def** callback\_f**(**downloaded\_size**,** block\_size**,** romote\_total\_size**):**

per **=** 100.0 **\*** downloaded\_size **\*** block\_size **/** romote\_total\_size

**if** per **>** 100**:**

per **=** 100

**print**"%.2f%%"**%** per

page **=** urllib2**.**urlopen**(**'http://dianying.fm/'**);**

soup **=** BeautifulSoup**(**page**)**

#打印页面的编码

**print** soup**.**originalEncoding

#打印规范化的html

**print** soup**.**prettify**().**decode**(**'utf-8'**)**

#打印title

titleTag**=**soup**.**html**.**head**.**title

**print** titleTag**.**string

#提取图片在本地目录

links**=**soup**.**findAll**(**'img'**)**

**for** i **in** links**:**

**if** str**(**i**[**'src'**])!=**''**:**

**print** str**(**i**).**decode**(**'utf-8'**)**

#print i['src']

urllib**.**urlretrieve**(**str**(**i**[**'src'**]),**"img/%s.jpg"**%(**uuid**.**uuid1**()),**callback\_f**)**

## 使用PyQuery来进行XML与Html的解析

在写PyQuery之前我一直都是使用BeautifulSoup来进行XML与Html的操作的，偶然的一个机会让我见到了PyQuery之后，真是非常喜爱，原因很简单PyQuery采用CSS选择器的方法来进行DOM元素操作，Jquery的技能可以有效继承过来。

PyQuery允许你使用jquery的方法来查询xml与html文件，它的API结构非常的接近jquery。PyQuery在内部使用lxml来操作xml与html所以速度是有保障的。

这是一个开源的项目，现在托管在Github上，大家可以自行查询到。

### 快速入门

你可以使用PyQuery封装类从一个字符串、一个lxml文档或者是一个文件还或者是一个url地址，反正来源地址可以非常丰富。

#coding:utf-8

**from** pyquery **import** PyQuery **as** pq

**from** lxml **import** etree

**import** urllib

d **=** pq**(**"<html></html>"**)**

d **=** pq**(**etree**.**fromstring**(**"<html></html>"**))**

d **=** pq**(**url**=**'http://google.com/'**)**

# d = pq(url='http://google.com/', opener=lambda url, \*\*kw: urllib.urlopen(url).read())

#path\_to\_html\_file='c:\\test.html'

#d = pq(filename=path\_to\_html\_file)

#coding:utf-8

**from** pyquery **import** PyQuery **as** pq

**from** lxml **import** etree

**import** urllib

d **=** pq**(**"<html></html>"**)**

d **=** pq**(**etree**.**fromstring**(**"<html></html>"**))**

d **=** pq**(**url**=**'http://2.myrestful.sinaapp.com/'**)**

# d = pq(url='http://google.com/', opener=lambda url, \*\*kw: urllib.urlopen(url).read())

#path\_to\_html\_file='c:\\test.html'

#d = pq(filename=path\_to\_html\_file)

**print** d**(**"#downloads"**)**

**print** '-----------'

p**=**d**(**"#downloads"**)**

#打印html

**print** p**.**html**()**

#打印text

**print** p**.**text**()**

### 元素操作

#coding:utf-8

**from** pyquery **import** PyQuery **as** pq

**from** lxml **import** etree

**import** urllib

p**=**pq**(**'<p id="hello" class="hello"></p>'**)**

**print** p**.**attr**(**"id"**)**

p**.**attr**(**"id"**,**"plop"**)**

**print** p**.**attr**(**"id"**)**

p**.**css**(**"font-size"**,**"15px"**)**

**print** p**.**attr**(**"style"**)**

#coding:utf-8

**from** pyquery **import** PyQuery **as** pq

**from** lxml **import** etree

**import** urllib

htmlsource**=**pq**(**url**=**'http://www.baidu.com'**)**

f**=**file**(**'out.html'**,**'w'**)**

f**.**write**(**htmlsource**.**html**().**encode**(**'utf-8'**))**

f**.**close**()**

### 其他操作

# coding:utf-8

**from** pyquery **import** PyQuery **as** pq

**from** lxml **import** etree

**import** urllib

p**=**pq**(**"<head><title>hello</title></head>"**)**

p**(**'head'**).**html**()**#返回<title>hello</title>

p**(**'head'**).**text**()**#返回hello

d**=**pq**(**'<div><p>test 1</p><p>test 2</p></div>'**)**

d**(**'p'**)**#返回[<p>,<p>]

**print** d**(**'p'**)**#返回<p>test 1</p><p>test 2</p>

**print** d**(**'p'**).**html**()**#返回test 1

**print** d**(**'p'**).**eq**(**1**).**html**()** #返回test 2

#filter() ——根据类名、id名得到指定元素

d**=**pq**(**"<div><p id='1'>test 1</p><p class='2'>test 2</p></div>"**)**

d**(**'p'**).**filter**(**'#1'**)** #返回[<p#1>]

d**(**'p'**).**filter**(**'.2'**)** #返回[<p.2>]

#find() ——查找嵌套元素

d**=**pq**(**"<div><p id='1'>test 1</p><p class='2'>test 2</p></div>"**)**

d**(**'div'**).**find**(**'p'**)**#返回[<p#1>, <p.2>]

d**(**'div'**).**find**(**'p'**).**eq**(**0**)**#返回[<p#1>]

#直接查找ID或者类名

d**=**pq**(**"<div><p id='1'>test 1</p><p class='2'>test 2</p></div>"**)**

d**(**'#1'**).**html**()**#返回test 1

d**(**'.2'**).**html**()**#返回test 2

#获取属性与修改属性

d**=**pq**(**"<p id='my\_id'><a href='http://hello.com'>hello</a></p>"**)**

d**(**'a'**).**attr**(**'href'**)**#返回http://hello.com

d**(**'p'**).**attr**(**'id'**)**#返回my\_id

d**(**'a'**).**attr**(**'href'**,** 'http://baidu.com'**)**

#addClass(value) ——为元素添加类

d**=**pq**(**'<div></div>'**)**

d**.**addClass**(**'my\_class'**)**#返回[<div.my\_class>]

#hasClass(name) #返回判断元素是否包含给定的类

d**=**pq**(**"<div class='my\_class'></div>"**)**

d**.**hasClass**(**'my\_class'**)**#返回True

#children(selector=None) ——获取子元素

d**=**pq**(**"<span><p id='1'>hello</p><p id='2'>world</p></span>"**)**

d**.**children**()**#返回[<p#1>, <p#2>]

d**.**children**(**'#2'**)**#返回[<p#2>]

#parents(selector=None)——获取父元素

d**=**pq**(**"<span><p id='1'>hello</p><p id='2'>world</p></span>"**)**

d**(**'p'**).**parents**()**#返回[<span>]

d**(**'#1'**).**parents**(**'span'**)**#返回[<span>]

d**(**'#1'**).**parents**(**'p'**)**#返回[]

#nextAll(selector=None) ——返回后面全部的元素块

d**=**pq**(**"<p id='1'>hello</p><p id='2'>world</p><img scr='' />"**)**

d**(**'p:first'**).**nextAll**()**#返回[<p#2>, <img>]

d**(**'p:last'**).**nextAll**()**#返回[<img>]

#not\_(selector) ——返回不匹配选择器的元素

d**=**pq**(**"<p id='1'>test 1</p><p id='2'>test 2</p>"**)**

d**(**'p'**).**not\_**(**'#2'**)**#返回[<p#1>]

# 操作Memcached

简单介绍：memcached很强大，它可以支持分布式的共享内存缓存，大型站点都用它。对小站点来说，有足够内存的话，使用它也可以得到超赞的效果。

## 安装

### Linux 环境

**安装包**

对于大多数Linux发行版本来说，可以使用官方推荐的方法：

Debian/Ubuntu

apt-get install memcached

Redhat/Fedora/CentOS

yum install memcached

### Win环境

1. 下载memcache的windows稳定版

下载地址 http://jehiah.cz/projects/memcached-win32/files/memcached-1.2.1-win32.zip

解压放某个盘下面，比如在c:\memcached

### linux启动memcached

启动参数说明：

-d 选项是启动一个守护进程

-m 是分配给Memcache使用的内存数量，单位是MB，默认64MB

-M return error on memory exhausted (rather than removing items)

-u 是运行Memcache的用户，如果当前为root 的话，需要使用此参数指定用户

-l 是监听的服务器IP地址，默认为所有网卡

-p 是设置Memcache的TCP监听的端口，最好是1024以上的端口

-c 选项是最大运行的并发连接数，默认是1024

-P 是设置保存Memcache的pid文件

-f chunk size growth factor (default: 1.25)

-I Override the size of each slab page. Adjusts max item size(1.4.2版本新增)

/usr/local/memcached/bin/memcached -d -m 100 -c 1000 -u root -p 11211

可以启动多个守护进程，但是端口不能重复。设置开机启动的话可以将上行命令增加到/etc/rc.d/rc.local文件中。

### windows下启动

在终端（也即cmd命令界面）cd到解压目录（这里是c:\memcached），运行 memcached.exe -d install 安装服务

运行memcached.exe -d start，memcached会使用默认的端口(11211)来启动，你可以在任务管理器中看到memcached.exe

-p 监听的端口

-l 连接的IP地址, 默认是本机

-d start 启动memcached服务

-d restart 重起memcached服务

-d stop|shutdown 关闭正在运行的memcached服务

-d install 安装memcached服务

-d uninstall 卸载memcached服务

-u 以的身份运行 (仅在以root运行的时候有效)

-m 最大内存使用，单位MB。默认64MB

-M 内存耗尽时返回错误，而不是删除项

-c 最大同时连接数，默认是1024

-f 块大小增长因子，默认是1.25

-n 最小分配空间，key+value+flags默认是48

-h 显示帮助

## Python操作Memcached

memcached API地址 http://code.google.com/p/memcached/wiki/Clients

网上流传说Python-API中效率最高的是python-libmemcached，这里居然看到了hongqn（豆瓣首席架构师，后来也得到证实python-libmemcached是豆瓣贡献），看来豆瓣的阳光真的是撒满了Python的各个角落。

另外还有python-memcached（100%纯Python），python-memcache（据说有内存泄漏问题？），cmemcache（代码多年未更新？）。

不过由于目前需要中效率并没有太高要求，于是选择了使用最多的python-memcached：

安装 python-memcached

easy\_install python-memcached

### Python操作memcached

import memcache

mc = memcache.Client(['127.0.0.1:11211'],debug=True)

mc.set('name','luo',60)

print mc.get('name')

### Memcached常用方法

memcache其实是一个map结构，最常用的几个函数：

保存数据

set(key,value,timeout) 把key映射到value，timeout指的是什么时候这个映射失效

add(key,value,timeout) 仅当存储空间中不存在键相同的数据时才保存

replace(key,value,timeout) 仅当存储空间中存在键相同的数据时才保存

获取数据

get(key) 返回key所指向的value

get\_multi([key1,key2,key3,key4]) 可以非同步地同时取得多个键值， 比循环调用get快数十倍

删除数据

delete(key, timeout) 删除键为key的数据，timeout为时间值，禁止在timeout时间内名为key的键保存新数据（set函数无效）。

# SQlite模块

## ****简单的介绍****

    SQLite数据库是一款非常小巧的嵌入式开源数据库软件，也就是说没有独立的维护进程，所有的维护都来自于程序本身。它是遵守ACID的关联式[数据库管理系统](http://baike.baidu.com/view/68446.htm)，它的设计目标是嵌入式的，而且目前已经在很多嵌入式产品中使用了它，它占用资源非常的低，在嵌入式设备中，可能只需要几百K的内存就够了。它能够支持Windows/Linux/Unix等等主流的[操作系统](http://baike.baidu.com/view/880.htm)，同时能够跟很多程序语言相结合，比如 Tcl、C#、PHP、Java等，还有ODBC接口，同样比起Mysql、PostgreSQL这两款开源世界著名的数据库管理系统来讲，它的处理速度比他们都快。SQLite第一个[Alpha版本](http://baike.baidu.com/view/707803.htm)诞生于2000年5月. 至今已经有10个年头，SQLite也迎来了一个版本 SQLite 3已经发布。

## ****安装与使用****

**1.导入Python SQLITE数据库模块**

    Python2.5之后，内置了SQLite3，成为了内置模块，这给我们省了安装的功夫，只需导入即可~

import sqlite3

**2. 创建/打开数据库**

   在调用connect函数的时候，指定库名称，如果指定的数据库存在就直接打开这个数据库，如果不存在就新创建一个再打开。

cx = sqlite3.connect("E:/test.db")

    也可以创建数据库在内存中。

con = sqlite3.connect(":memory:")

**3.数据库连接对象**

   打开数据库时返回的对象cx就是一个数据库连接对象，它可以有以下操作：

* commit()--事务提交
* rollback()--事务回滚
* close()--关闭一个数据库连接
* cursor()--创建一个游标

   关于commit()，如果isolation\_level隔离级别默认，那么每次对数据库的操作，都需要使用该命令，你也可以设置isolation\_level=None，这样就变为自动提交模式。

**4.使用游标查询数据库**

    我们需要使用游标对象SQL语句查询数据库，获得查询对象。 通过以下方法来定义一个游标。

cu=cx.cursor()

* 游标对象有以下的操作：
* execute()--执行sql语句
* executemany--执行多条sql语句
* close()--关闭游标
* fetchone()--从结果中取一条记录，并将游标指向下一条记录
* fetchmany()--从结果中取多条记录
* fetchall()--从结果中取出所有记录
* scroll()--游标滚动

**1. 建表**

cu.execute("create table catalog (id integer primary key,pid integer,name varchar(10) UNIQUE,nickname text NULL)")

上面语句创建了一个叫catalog的表，它有一个主键id，一个pid，和一个name，name是不可以重复的，以及一个nickname默认为NULL。

**2. 插入数据**

请注意避免以下写法：

# Never do this -- insecure 会导致注入攻击

pid=200

c.execute("... where pid = '%s'" % pid)

正确的做法如下，如果t只是单个数值，也要采用t=(n,)的形式，因为元组是不可变的。

for t in[(0,10,'abc','Yu'),(1,20,'cba','Xu')]:

cx.execute("insert into catalog values (?,?,?,?)", t)

简单的插入两行数据,不过需要提醒的是,只有提交了之后,才能生效.我们使用数据库连接对象cx来进行提交commit和回滚rollback操作.

cx.commit()

**3.查询**

cu.execute("select \* from catalog")

要提取查询到的数据,使用游标的fetch函数,如:

In [10]: cu.fetchall()

Out[10]: [(0, 10, u'abc', u'Yu'), (1, 20, u'cba', u'Xu')]

如果我们使用cu.fetchone(),则首先返回列表中的第一项,再次使用,则返回第二项,依次下去.

**4.修改**

In [12]: cu.execute("update catalog set name='Boy' where id = 0")

In [13]: cx.commit()

注意,修改数据以后提交

**5.删除**

cu.execute("delete from catalog where id = 1")

cx.commit()

**6.使用中文**

请先确定你的IDE或者系统默认编码是utf-8,并且在中文前加上u

x=u'鱼'

cu.execute("update catalog set name=? where id = 0",x)

cu.execute("select \* from catalog")

cu.fetchall()

[(0, 10, u'\u9c7c', u'Yu'), (1, 20, u'cba', u'Xu')]

如果要显示出中文字体，那需要依次打印出每个字符串

In [26]: for item in cu.fetchall():  
   ....:     for element in item:  
   ....:         print element,  
   ....:     print  
   ....:

0 10 鱼 Yu

1 20 cba Xu

游标提供了一种对从表中检索出的数据进行操作的灵活手段，就本质而言，游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标总是与一条SQL 选择语句相关联。因为游标由结果集（可以是零条、一条或由相关的选择语句检索出的多条记录）和结果集中指向特定记录的游标位置组成。当决定对结果集进行处理时，必须声明一个指向该结果集的游标。如果曾经用 C 语言写过对文件进行处理的程序，那么游标就像您打开文件所得到的文件句柄一样，只要文件打开成功， 该文件句柄就可代表该文件。对于游标而言，其道理是相同的。可见游标能够实现按与传统程序读取平面文件类似的方式处理来自基础表的结果集，从而把表中数据以平面文件的形式呈现给程序。

我们知道关系数据库管理系统实质是面向集合的，在Sqlite中并没有一种描述表中单一记录的表达形式，除非使用where 子句来限制只有一条记录被选中。因此我们必须借助于游标来进行面向单条记录的数据处理。由此可见，游标允许应用程序对查询语句select 返回的行结果集中每一行进行相同或不同的操作，而不是一次对整个结果集进行同一种操作；它还提供对基于游标位置而对表中数据进行删除或更新的能力；正是游标把作为面向集合的数据库管理系统和面向行的程序设计两者联系起来，使两个数据处理方式能够进行沟通。

## 一个简单的例子

#coding:utf-8

**import** sqlite3

**import** sys

**def** sqlite\_basic**():**

# Connect to db

conn **=** sqlite3**.**connect**(**'test1.db'**)**

# create cursor

c **=** conn**.**cursor**()**

# Create table

c**.**execute**(**'''

create table if not exists stocks

(date text, trans text, symbol text,

qty real, price real)

'''

**)**

# Insert a row of data

c**.**execute**(**'''

insert into stocks

values ('2006-01-05','BUY','楚',100,35.14)

'''

**)**

# query the table

rows **=** c**.**execute**(**"select \* from stocks"**)**

# print the table

**for** row **in** rows**:**

**print(**row**)**

# delete the row

c**.**execute**(**"delete from stocks where symbol=='REHT'"**)**

# Save (commit) the changes

conn**.**commit**()**

# Close the connection

conn**.**close**()**

**def** sqlite\_adv**():**

conn **=** sqlite3**.**connect**(**'test2.db'**)**

c **=** conn**.**cursor**()**

c**.**execute**(**'''

create table if not exists employee

(id text, name text, age inteage)

'''**)**

# insert many rows

**for** t **in** **[(**'1'**,** 'itech'**,** 10**),**

**(**'2'**,** 'jason'**,** 10**),**

**(**'3'**,** 'jack'**,** 30**),**

**]:**

c**.**execute**(**'insert into employee values (?,?,?)'**,** t**)**

# create index

create\_index **=** 'CREATE INDEX IF NOT EXISTS idx\_id ON employee (id);'

c**.**execute**(**create\_index**)**

# more secure

t **=** **(**'jason'**,)**

c**.**execute**(**'select \* from employee where name=?'**,** t**)**

# fetch query result

**for** row **in** c**.**fetchall**():**

**print(**row**)**

conn**.**commit**()**

conn**.**close**()**

**def** sqlite\_adv2**():**

reload**(**sys**)**

sys**.**setdefaultencoding**(**'utf-8'**)**

# memory db

con **=** sqlite3**.**connect**(**":memory:"**)**

cur **=** con**.**cursor**()**

# execute sql

cur**.**executescript**(**'''

create table book(

title,

author,

published

);

insert into book(title, author, published)

values (

'AAA book',

'Douglas Adams',"kkk");

'''**)**

rows **=** cur**.**execute**(**"select \* from book"**)**

**for** row **in** rows**:**

**print(**"title:" **+** row**[**0**])**

**print(**"author:" **+** row**[**1**])**

**print(**"published:" **+** str**(**row**[**2**]))**

**def** sqlite\_adv3**():**

**import** datetime

# Converting SQLite values to custom Python types

# Default adapters and converters for datetime and timestamp

con **=** sqlite3**.**connect**(**":memory:"**,** detect\_types**=**sqlite3**.**PARSE\_DECLTYPES **|** sqlite3**.**PARSE\_COLNAMES**)**

cur **=** con**.**cursor**()**

cur**.**execute**(**"create table test(d date, ts timestamp)"**)**

today **=** datetime**.**date**.**today**()**

now **=** datetime**.**datetime**.**now**()**

cur**.**execute**(**"insert into test(d, ts) values (?, ?)"**,** **(**today**,** now**))**

cur**.**execute**(**"select d, ts from test"**)**

row **=** cur**.**fetchone**()**

**print** today**,** "=>"**,** row**[**0**],** type**(**row**[**0**])**

**print** now**,** "=>"**,** row**[**1**],** type**(**row**[**1**])**

cur**.**execute**(**'select current\_date as "d [date]", current\_timestamp as "ts [timestamp]" from test'**)**

row **=** cur**.**fetchone**()**

**print** "current\_date"**,** row**[**0**],** type**(**row**[**0**])**

**print** "current\_timestamp"**,** row**[**1**],** type**(**row**[**1**])**

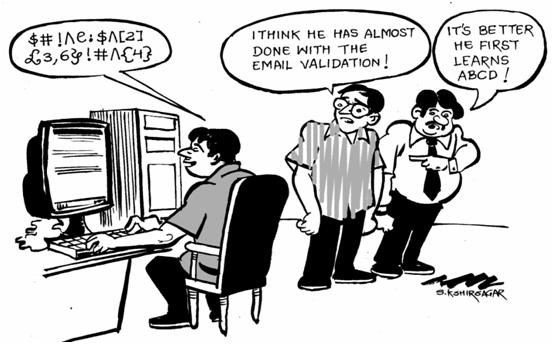
sqlite\_basic**()**

## 中文处理

# 正则表达式

## 简介

编写验证规则最流行和最简单的方法就是正则表达式了，但唯一的一个问题是正则表达式的语法太隐晦了，让人蛋疼无比。很多开发者为了在项目中应用复杂的验证，经常要使用一些小抄来记住正则式的复杂语法和各种常用命令。



也许你是初学者，那以防万一，我先来讲讲什么是正则表达式吧：

正则表达式可以帮助我们更好的描述复杂的文本格式。一旦你描述清楚了这些格式，那你就可以利用它们对文本数据进行检索、替换、提取和修改操作。

下面有一个正则表达式的简单例子。第一步先要引入有关正则式的包：

# -\*- coding: utf-8 -\*-

**import** re

第二步就是用指定的正则式构建一个正则表达式对象,下面的正则式是用来搜索长度为10的a-z的英文字母:

# -\*- coding: utf-8 -\*-

**import** re

regex **=** re**.**compile**(**'[a-z]{10}'**)**

最后,根据正则式在指定数据中检索匹配项,如果匹配IsMatch方法就会返回true。

# -\*- coding: utf-8 -\*-

**import** re

regex **=** re**.**compile**(**'[a-z]{10}'**)**

m1 **=** re**.**match**(**regex**,** '1234567890'**)**

m2 **=** re**.**match**(**regex**,** 'abcdefghig'**)**

# m = regex.search("test1234.567").group()

**if** m1 **is** **not** **None:**

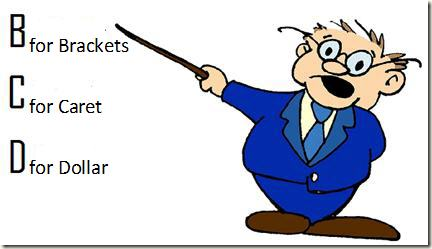
**print** 'True'

**if** m2 **is** **not** **None:**

**print** 'True'

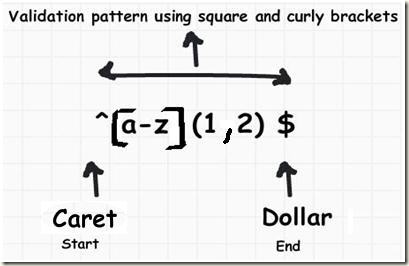
## 3个重要的正则式命令

记住正则语法最好的办法就是记住这三样东西：Bracket（括号）, caret（插入符号）和Dollars（美元符号）。



|  |  |  |
| --- | --- | --- |
| B（括号） | | 在正则表达式中有3种类型的括号  方括号 “[“和花括号“{“ 。  方括号"["内是需要匹配的字符，花括号"{"内是指定匹配字符的数量。  圆括号“(“ 则是用来分组的。 |
| C（插入符号） | 插入符号 “**^**” 表示正则式的开始。 | |
| D（美元符号） | 美元符号“**$**” 表示正则式的结束。 | |

现在你知道上面的3个语法后，你就可以写世界上任何一条验证规则了。比如下面的例子就很好的说明了上面3条正则语法是如何协调运作的。



（注：上图有个错误，"()"应为"{}"）

* 上面的这条正则式只能匹配a-z的英文字母，同样是在中括号中标明匹配范围。
* 花括号中则是标明匹配字符串的最小长度和最大长度。
* 最后为了让表达式更规则，分别在开头和结尾加上了插入符号"^"和美元符号"$"。

### 验证字符串

好了，现在我们就用上面的3条语法来实现一些正则表达式的验证规则吧。

检查用户是否输入了shivkoirala？

shivkoirala

让我们开始第一个验证，输入的字符在a-g之间？

[a-g]

输入的字符在a-g之间并且长度为3？

[a-g]{3}

输入的字符在a-g之间并且最大长度为3最小长度为1？

[a-g]{1,3}

我如何在匹配像91230456, 01237648那样的固定8位数？

^[0-9]{8}$

如何验证最小长度为3最大长度为7的数字，如：123, 1274667, 87654？

^[0-9]{3,7}$

 如何验证像LJI10201020那样的发票编号，前3个是字母剩余为8位长度的数字？前三个是字母：

^[a-z]{3}

后面是8位长度的数字：

[0-9]{8}

所以整个表达式为：

^[a-z]{3}[0-9]{8}$

验证像INV19020303 或 inv82083003那样的前3位是不区分大小写的英文字母，剩余8位是数字在前面的表达式中只能匹配前3个是小写英文字母的发票编号，如果我们输入大写字母那就不能匹配了。所以为了确保前3个字母是不区分大小写的，我们就要用表达式^[a-zA-Z]{3}。

完整的正则式如下：

^[a-zA-Z]{3}[0-9]{8}$

### 验证url

我们可以验证简单的网址URL格式吗？

第一步：检查是否存在www：

^www.

第二步：域名必须是长度在1-15的英文字母：

.[a-z]{1,15}

第三部：以.com或者.org结束：

.(com|org)$

完整的表达式如下：

^www[.][a-z]{1,15}[.](com|org)$

### 验证Email

让我们在来看看BCD（其实也就是上面说的3条基本语法）如何验证email格式

第一步：email开始是长度在1-10的英文字母，最后跟一个"@"：

^[a-zA-Z0-9]{1,10}@

第二步：@后面是长度在1-10的英文字母，后面跟一个"."：

[a-zA-Z]{1,10}.

第三步：最后以.com或.org结束：

.(com|org)$

最后完整的表达式如下：

^[a-zA-Z0-9]{1,10}@[a-zA-Z]{1,10}.(com|org)$

### 验证值在0-25的数字

^(([0-9])|([0-1][0-9])|([0-2][0-5]))$

### 验证格式为MM/DD/YYYY, YYYY/MM/DD and DD/MM/YYYY的日期

|  |  |  |
| --- | --- | --- |
| 步骤 | 正则式 | 描述说明 |
| 先来检查 DD. 首先DD的长度为1-29 ( 2月份) , 1-30 (月小) , 1-31 (月大) .  所以 DD 就是 1-9 或 01-09 | [1-9]|0[1-9] | 允许用户输入1-9或者01-09. |
| 再为DD添加匹配10-19 | [1-9]|1[0-9] | 允许用户输入01-19. |
| 再为DD添加匹配20-29 | [1-9]|1[0-9]|2[0-9] | 允许用户输入01-29. |
| i再为DD添加匹配30-31 | [1-9]|1[0-9]|2[0-9]|3[0-1] | 最后用户可以输入01-31. |
| 再来匹配日期间的分隔符"/"，"-" | [/ . -] | 允许用户输入日期分隔符. |
| MM也是类似的操作 | [1-9]|0[1-9]|1[0-2] | 让用户输入月份值01-12. |
| 最后就是YY的操作 | 1[9][0-9][0-9]|2[0][0-9][0-9] | 允许用户输入年份1900-2099. |

最后DD/MM/YYYY格式的日期的正则表达式为：

^([1-9]|0[1-9]|1[0-9]|2[0-9]|3[0-1])[- / .]([1-9]|0[1-9]|1[0-2])[- / .](1[9][0-9][0-9]|2[0][0-9][0-9])$

MM/DD/YYYY格式的日期：

^([1-9]|0[1-9]|1[0-2])[- / .]([1-9]|0[1-9]|1[0-9]|2[0-9]|3[0-1])[- / .](1[9][0-9][0-9]|2[0][0-9][0-9])$

YYYY/MM/DD格式的日期：

^(1[9][0-9][0-9]|2[0][0-9][0-9])[- / .]([1-9]|0[1-9]|1[0-2])[- / .]([1-9]|0[1-9]|1[0-9]|2[0-9]|3[0-1])$

## 正则表达符

### 你常用的元字符

|  |  |
| --- | --- |
| **代码** | **说明** |
| . | 匹配除换行符以外的任意字符 |
| \w | 匹配字母或数字或下划线或汉字 |
| \s | 匹配任意的空白符 |
| \d | 匹配数字 |
| \b | 匹配单词的开始或结束 |
| ^ | 匹配字符串的开始 |
| $ | 匹配字符串的结束 |

比如一个网站如果要求你填写的QQ号必须为5位到12位数字时，可以使用：

^\d{5,12}$

### 常用的限定符

|  |  |
| --- | --- |
| **代码** | **说明** |
| \* | 重复零次或更多次 |
| + | 重复一次或更多次 |
| ? | 重复零次或一次 |
| {n} | 重复n次 |
| {n,} | 重复n次或更多次 |
| {n,m} | 重复n到m次 |

下面是一些使用重复的例子：

Windows\d+匹配Windows后面跟1个或更多数字

^\w+匹配一行的第一个单词(或整个字符串的第一个单词，具体匹配哪个意思得看选项设置)

^:匹配字符串的开始

$:匹配字符串的结尾

\b:匹配一个单词的边界

\d:匹配任意数字

\D:匹配任意非数字字符

x?:匹配一个可选的x字符（换句话说，它匹配1次或者0次x 字符）

x\*:匹配0次或者多次x字符

x+:匹配1次或者多次x字符

x{n,m}: 匹配x字符，至少n次，至多m次

{a|b|c}:要么匹配a，要么匹配b，要么匹配c

(x):一般情况下表示一个记忆组(remembered group)

你可以利用re.search函数返回对象的groups()函数获取它的值

### 字符串类

要想查找数字，字母或数字，空白是很简单的，因为已经有了对应这些字符集合的元字符，但是如果你想匹配没有预定义元字符的字符集合(比如元音字母a,e,i,o,u),应该怎么办？

很简单，你只需要在方括号里列出它们就行了，像[aeiou]就匹配任何一个英文元音字母，[.?!]匹配标点符号(.或?或!)。

我们也可以轻松地指定一个字符范围，像[0-9]代表的含意与\d就是完全一致的：一位数字；同理[a-z0-9A-Z\_]也完全等同于\w（如果只考虑英文的话）。

如果我们要比配一个电话区号：像(010)88886666，或022-22334455，或02912345678等

第一步：前面的区号可能有括号也可能没有，区号与电话号码之间可能有-号也可能没有

首先是一个转义字符\(,它能出现0次或1次?

然后是一个0，后面跟着2个数字\d{2}

然后是)或-或空格中的一个，它出现1次或不出现?，最后是8个数字\d{8}

\(?0\d{2}[)-]

第二步：匹配后面的电话

\(?0\d{2}[)-]?\d{8}

“(”和“)”也是元字符，后面的分组节里会提到，所以在这里需要使用转义。

这个表达式可以匹配几种格式的电话号码，像(010)88886666，或022-22334455，或02912345678等。我们对它进行一些分析吧：首先是一个转义字符\(,它能出现0次或1次(?),然后是一个0，后面跟着2个数字\d{2}，然后是)或-或空格中的一个，它出现1次或不出现(?)，最后是8个数字(\d{8})。

# -\*- coding: utf-8 -\*-

**import** re

regex **=** re**.**compile**(**'\(?0\d{2}[)-]?\d{8}'**)**

m1 **=** re**.**search**(**regex**,** '(010)88886666'**)**

m2 **=** re**.**search**(**regex**,** '022-22334455'**)**

m3 **=** re**.**search**(**regex**,** '02912345678'**)**

m4 **=** re**.**search**(**regex**,** '(010)88886666 022-22334455 02912345678'**)**

**if** m4 **is** **not** **None:**

**print** m4**.**group**()**

m5 **=** re**.**findall**(**regex**,** '(010)88886666 022-2233s4455 02912345678'**)**

**print** m5

### 分枝条件

不幸的是，刚才那个表达式也能匹配010)12345678或(022-87654321这样的“不正确”的格式。

要解决这个问题，我们需要用到分枝条件。正则表达式里的分枝条件指的是有几种规则，如果满足其中任意一种规则都应该当成匹配，具体方法是用|把不同的规则分隔开。听不明白？没关系，看例子：

0\d{2}-\d{8}|0\d{3}-\d{7}这个表达式能匹配两种以连字号分隔的电话号码：一种是三位区号，8位本地号如010-12345678，一种是4位区号，7位本地号0376-2233445

\(0\d{2}\)[- ]?\d{8}|0\d{2}[- ]?\d{8}这个表达式匹配3位区号的电话号码，其中区号可以用小括号括起来，也可以不用，区号与本地号间可以用连字号或空格间隔，也可以没有间隔。你可以试试用分枝条件把这个表达式扩展成也支持4位区号的。

\d{5}-\d{4}|\d{5}这个表达式用于匹配美国的邮政编码。美国邮编的规则是5位数字，或者用连字号间隔的9位数字。之所以要给出这个例子是因为它能说明一个问题：使用分枝条件时，要注意各个条件的顺序。如果你把它改成\d{5}|\d{5}-\d{4}的话，那么就只会匹配5位的邮编(以及9位邮编的前5位)。原因是匹配分枝条件时，将会从左到右地测试每个条件，如果满足了某个分枝的话，就不会去再管其它的条件了。

### 分组

我们已经提到了怎么重复单个字符（直接在字符后面加上限定符就行了）；但如果想要重复多个字符又该怎么办？你可以用小括号来指定**子表达式**(也叫做**分组**)，然后你就可以指定这个子表达式的重复次数了，你也可以对子表达式进行其它一些操作(后面会有介绍)。

(\d{1,3}\.){3}\d{1,3}是一个简单的IP地址匹配表达式。要理解这个表达式，请按下列顺序分析它：

\d{1,3}匹配1到3位的数字

(\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个**分组**)重复3次

最后再加上一个一到三位的数字(\d{1,3})

IP地址中每个数字都不能大于255

不幸的是，它也将匹配*256.300.888.999*这种不可能存在的IP地址。如果能使用算术比较的话，或许能简单地解决这个问题，但是正则表达式中并不提供关于数学的任何功能，所以只能使用冗长的分组，选择，字符类来描述一个正确的IP地址：

((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)

理解这个表达式的关键是理解2[0-4]\d|25[0-5]|[01]?\d\d?，这里我就不细说了，你自己应该能分析得出来它的意义。

### 反义

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外，其它任意字符都行的情况，这时需要用到**反义**：

|  |  |
| --- | --- |
| **表3.常用的反义代码** | |
| **代码/语法** | **说明** |
| \W | 匹配任意不是字母，数字，下划线，汉字的字符 |
| \S | 匹配任意不是空白符的字符 |
| \D | 匹配任意非数字的字符 |
| \B | 匹配不是单词开头或结束的位置 |
| [^x] | 匹配除了x以外的任意字符 |
| [^aeiou] | 匹配除了aeiou这几个字母以外的任意字符 |

例子：

\S+匹配不包含空白符的字符串

<a[^>]+>匹配用尖括号括起来的以a开头的字符串

### 后向引用

使用小括号指定一个子表达式后，**匹配这个子表达式的文本**(也就是此分组捕获的内容)可以在表达式或其它程序中作进一步的处理。默认情况下，每个分组会自动拥有一个**组号**，规则是：从左向右，以分组的左括号为标志，第一个出现的分组的组号为1，第二个为2，以此类推。

* 分组0对应整个正则表达式
* 实际上组号分配过程是要从左向右扫描两遍的：第一遍只给未命名组分配，第二遍只给命名组分配－－因此所有命名组的组号都大于未命名的组号
* 你可以使用(?:exp)这样的语法来剥夺一个分组对组号分配的参与权．

**后向引用**用于重复搜索前面某个分组匹配的文本。例如，\1代表分组1匹配的文本。难以理解？请看示例：

\b(\w+)\b\s+\1\b可以用来匹配重复的单词，像*go go*, 或者*kitty kitty*。这个表达式首先是一个单词，也就是单词开始处和结束处之间的多于一个的字母或数字(\b(\w+)\b)，这个单词会被捕获到编号为1的分组中，然后是1个或几个空白符(\s+)，最后是分组1中捕获的内容（也就是前面匹配的那个单词）(\1)。

你也可以自己指定子表达式的**组名**。要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+)(或者把尖括号换成'也行：(?'Word'\w+)),这样就把\w+的组名指定为Word了。要反向引用这个分组**捕获**的内容，你可以使用\k<Word>,所以上一个例子也可以写成这样：\b(?<Word>\w+)\b\s+\k<Word>\b。

使用小括号的时候，还有很多特定用途的语法。下面列出了最常用的一些：

|  |  |  |
| --- | --- | --- |
| **表4.常用分组语法** | | |
| **分类** | **代码/语法** | **说明** |
| **捕获** | (exp) | 匹配exp,并捕获文本到自动命名的组里 |
| (?<name>exp) | 匹配exp,并捕获文本到名称为name的组里，也可以写成(?'name'exp) |
| (?:exp) | 匹配exp,不捕获匹配的文本，也不给此分组分配组号 |
| **零宽断言** | (?=exp) | 匹配exp前面的位置 |
| (?<=exp) | 匹配exp后面的位置 |
| (?!exp) | 匹配后面跟的不是exp的位置 |
| (?<!exp) | 匹配前面不是exp的位置 |
| **注释** | (?#comment) | 这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读 |

我们已经讨论了前两种语法。第三个(?:exp)不会改变正则表达式的处理方式，只是这样的组匹配的内容不会像前两种那样被捕获到某个组里面，也不会拥有组号。“我为什么会想要这样做？”——好问题，你觉得为什么呢？

### 贪婪与懒惰

当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配**尽可能多**的字符。以这个表达式为例：a.\*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索*aabab*的话，它会匹配整个字符串aabab。这被称为**贪婪**匹配。

有时，我们更需要**懒惰**匹配，也就是匹配**尽可能少**的字符。前面给出的限定符都可以被转化为懒惰匹配模式，只要在它后面加上一个问号?。这样.\*?就意味着匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。现在看看懒惰版的例子吧：

a.\*?b匹配最短的，以a开始，以b结束的字符串。如果把它应用于*aabab*的话，它会匹配aab（第一到第三个字符）和ab（第四到第五个字符）。

为什么第一个匹配是aab（第一到第三个字符）而不是ab（第二到第三个字符）？简单地说，因为正则表达式有另一条规则，比懒惰／贪婪规则的优先级更高：最先开始的匹配拥有最高的优先权——The match that begins earliest wins。

|  |  |
| --- | --- |
| **表5.懒惰限定符** | |
| **代码/语法** | **说明** |
| \*? | 重复任意次，但尽可能少重复 |
| +? | 重复1次或更多次，但尽可能少重复 |
| ?? | 重复0次或1次，但尽可能少重复 |
| {n,m}? | 重复n到m次，但尽可能少重复 |
| {n,}? | 重复n次以上，但尽可能少重复 |

## 在Python中使用

Python 自1.5版本起增加了re 模块，它提供 Perl 风格的正则表达式模式。Python 1.5之前版本则是通过 regex 模块提供 Emacs 风格的模式。Emacs 风格模式可读性稍差些，而且功能也不强，因此编写新代码时尽量不要再使用 regex 模块，当然偶尔你还是可能在老代码里发现其踪影。

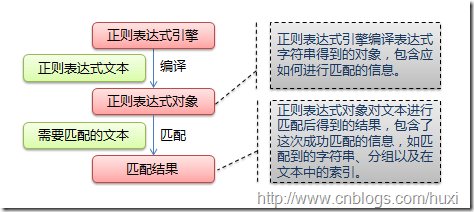
就其本质而言，正则表达式（或 RE）是一种小型的、高度专业化的编程语言，（在Python中）它内嵌在Python中，并通过 re 模块实现。使用这个小型语言，你可以为想要匹配的相应字符串集指定规则；该字符串集可能包含英文语句、e-mail地址、TeX命令或任何你想搞定的东西。然后你可以问诸如“这个字符串匹配该模式吗？”或“在这个字符串中是否有部分匹配该模式呢？”。你也可以使用 RE 以各种方式来修改或分割字符串。

正则表达式模式被编译成一系列的字节码，然后由用 C 编写的匹配引擎执行。在高级用法中，也许还要仔细留意引擎是如何执行给定 RE ，如何以特定方式编写 RE 以令生产的字节码运行速度更快。本文并不涉及优化，因为那要求你已充分掌握了匹配引擎的内部机制。

正则表达式语言相对小型和受限（功能有限），因此并非所有字符串处理都能用正则表达式完成。当然也有些任务可以用正则表达式完成，不过最终表达式会变得异常复杂。碰到这些情形时，编写 Python 代码进行处理可能反而更好；尽管 Python 代码比一个精巧的正则表达式要慢些，但它更易理解。

正则表达式并不是Python的一部分。正则表达式是用于处理字符串的强大工具，拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如str自带的方法，但功能十分强大。得益于这一点，在提供了正则表达式的语言里，正则表达式的语法都是一样的，区别只在于不同的编程语言实现支持的语法数量不同；但不用担心，不被支持的语法通常是不常用的部分。如果已经在其他语言里使用过正则表达式，只需要简单看一看就可以上手了。

下图展示了使用正则表达式进行匹配的流程：



正则表达式的大致匹配过程是：依次拿出表达式和文本中的字符比较，如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。如果表达式中有量词或边界，这个过程会稍微有一些不同，但也是很好理解的，看下图中的示例以及自己多使用几次就能明白。

下图列出了Python支持的正则表达式元字符和语法：





## Re模块

Python通过re模块提供对正则表达式的支持。使用re的一般步骤是先将正则表达式的字符串形式编译为Pattern实例，然后使用Pattern实例处理文本并获得匹配结果（一个Match实例），最后使用Match实例获得信息，进行其他的操作。

# encoding: UTF-8

**import** re

# 将正则表达式编译成Pattern对象

pattern **=** re**.**compile**(**r'hello'**)**

# 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None

match **=** pattern**.**match**(**'hello world!'**)**

**if** match**:**

# 使用Match获得分组信息

**print** match**.**group**()**

### 输出 ###

# hello

**re.compile(strPattern[, flag]):**

这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。 第二个参数flag是匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如re.I | re.M。另外，你也可以在regex字符串中指定模式，比如re.compile('pattern', re.I | re.M)与re.compile('(?im)pattern')是等价的。 可选值有：

* re.**I**(re.IGNORECASE): 忽略大小写（括号内是完整写法，下同）
* **M**(MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
* **S**(DOTALL): 点任意匹配模式，改变'.'的行为
* **L**(LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
* **U**(UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于unicode定义的字符属性
* **X**(VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。以下两个正则表达式是等价的：

# encoding: UTF-8

import re

a = re.compile(r"""\d + # the integral part

\. # the decimal point

\d \* # some fractional digits""", re.X)

b = re.compile(r"\d+\.\d\*")

re提供了众多模块方法用于完成正则表达式的功能。这些方法可以使用Pattern实例的相应方法替代，唯一的好处是少写一行re.compile()代码，但同时也无法复用编译后的Pattern对象。这些方法将在Pattern类的实例方法部分一起介绍。如上面这个例子可以简写为：

m = re.match(r'hello', 'hello world!')

print m.group()

re模块还提供了一个方法escape(string)，用于将string中的正则表达式元字符如\*/+/?等之前加上转义符再返回，在需要大量匹配元字符时有那么一点用。

## Match方法

Match对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用Match提供的可读属性或方法来获取这些信息。

属性：

* string: 匹配时使用的文本。
* **re**: 匹配时使用的Pattern对象。
* **pos**: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
* **endpos**: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
* **lastindex**: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为None。
* **lastgroup**: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为None。

方法：

* **group([group1, …]):**  
  获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
* **groups([default]):**    
  以元组形式返回全部分组截获的字符串。相当于调用group(1,2,…last)。default表示没有截获字符串的组以这个值替代，默认为None。
* **groupdict([default]):**返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
* **start([group]):**   
  返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
* **end([group]):**返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。
* **span([group]):**返回(start(group), end(group))。
* **expand(template):**   
  将匹配到的分组代入template中然后返回。template中可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。\id与\g<id>是等价的；但\10将被认为是第10个分组，如果你想表达\1之后是字符'0'，只能使用\g<1>0。

# encoding: UTF-8

**import** re

m **=** re**.**match**(**r'(\w+) (\w+)(?P<sign>.\*)'**,** 'hello world! sign'**)**

**print** "m.string:"**,** m**.**string

**print** "m.re:"**,** m**.**re

**print** "m.pos:"**,** m**.**pos

**print** "m.endpos:"**,** m**.**endpos

**print** "m.lastindex:"**,** m**.**lastindex

**print** "m.lastgroup:"**,** m**.**lastgroup

**print** "m.group(1,2):"**,** m**.**group**(**1**,** 2**)**

**print** "m.groups():"**,** m**.**groups**()**

**print** "m.groupdict():"**,** m**.**groupdict**()**

**print** "m.start(2):"**,** m**.**start**(**2**)**

**print** "m.end(2):"**,** m**.**end**(**2**)**

**print** "m.span(2):"**,** m**.**span**(**2**)**

**print** r"m.expand(r'\2 \1\3'):"**,** m**.**expand**(**r'\2 \1\3'**)**

### output ###

# m.string: hello world!

# m.re: <\_sre.SRE\_Pattern object at 0x016E1A38>

# m.pos: 0

# m.endpos: 12

# m.lastindex: 3

# m.lastgroup: sign

# m.group(1,2): ('hello', 'world')

# m.groups(): ('hello', 'world', '!')

# m.groupdict(): {'sign': '!'}

# m.start(2): 6

# m.end(2): 11

# m.span(2): (6, 11)

# m.expand(r'\2 \1\3'): world hello!

## Pattern方法

Pattern对象是一个编译好的正则表达式，通过Pattern提供的一系列方法可以对文本进行匹配查找。

Pattern不能直接实例化，必须使用re.compile()进行构造。

Pattern提供了几个可读属性用于获取表达式的相关信息：

* pattern: 编译时用的表达式字符串。
* flags: 编译时用的匹配模式。数字形式。
* groups: 表达式中分组的数量。
* groupindex: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。

# encoding: UTF-8

import re

p = re.compile(r'(\w+) (\w+)(?P<sign>.\*)', re.DOTALL)

print "p.pattern:", p.pattern

print "p.flags:", p.flags

print "p.groups:", p.groups

print "p.groupindex:", p.groupindex

### output ###

# p.pattern: (\w+) (\w+)(?P<sign>.\*)

# p.flags: 16

# p.groups: 3

# p.groupindex: {'sign': 3}

实例方法[ | re模块方法]：

* **match(string[, pos[, endpos]]) | re.match(pattern, string[, flags]):**这个方法将从string的pos下标处起尝试匹配pattern；如果pattern结束时仍可匹配，则返回一个Match对象；如果匹配过程中pattern无法匹配，或者匹配未结束就已到达endpos，则返回None。   
  pos和endpos的默认值分别为0和len(string)；re.match()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。   
  注意：这个方法并不是完全匹配。当pattern结束时若string还有剩余字符，仍然视为成功。想要完全匹配，可以在表达式末尾加上边界匹配符'$'。
* **search(string[, pos[, endpos]]) | re.search(pattern, string[, flags]):**这个方法用于查找字符串中可以匹配成功的子串。从string的pos下标处起尝试匹配pattern，如果pattern结束时仍可匹配，则返回一个Match对象；若无法匹配，则将pos加1后重新尝试匹配；直到pos=endpos时仍无法匹配则返回None。   
  pos和endpos的默认值分别为0和len(string))；re.search()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

# encoding: UTF-8

import re

# 将正则表达式编译成Pattern对象

pattern = re.compile(r'world')

# 使用search()查找匹配的子串，不存在能匹配的子串时将返回None

# 这个例子中使用match()无法成功匹配

match = pattern.search('hello world!')

if match:

# 使用Match获得分组信息

print match.group()

### 输出 ###

# world

**split(string[, maxsplit]) | re.split(pattern, string[, maxsplit]):**按照能够匹配的子串将string分割后返回列表。maxsplit用于指定最大分割次数，不指定将全部分割。

import re

p = re.compile(r'\d+')

print p.split('one1two2three3four4')

### output ###

# ['one', 'two', 'three', 'four', '']

**findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags]):**搜索string，以列表形式返回全部能匹配的子串。

# encoding: UTF-8

import re

p = re.compile(r'\d+')

print p.findall('one1two2three3four4')

### output ###

# ['1', '2', '3', '4']

**finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags]):**搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

# encoding: UTF-8

import re

p = re.compile(r'\d+')

for m in p.finditer('one1two2three3four4'):

print m.group(),

### output ###

# 1 2 3 4

**sub(repl, string[, count]) | re.sub(pattern, repl, string[, count]):**使用repl替换string中每一个匹配的子串后返回替换后的字符串。   
当repl是一个字符串时，可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。   
当repl是一个方法时，这个方法应当只接受一个参数（Match对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。   
count用于指定最多替换次数，不指定时全部替换。

# encoding: UTF-8

import re

p = re.compile(r'(\w+) (\w+)')

s = 'i say, hello world!'

print p.sub(r'\2 \1', s)

def func(m):

return m.group(1).title() + ' ' + m.group(2).title()

print p.sub(func, s)

### output ###

# say i, world hello!

# I Say, Hello World!

**subn(repl, string[, count]) |re.sub(pattern, repl, string[, count]):**返回 (sub(repl, string[, count]), 替换次数)。

# encoding: UTF-8

import re

p = re.compile(r'(\w+) (\w+)')

s = 'i say, hello world!'

print p.subn(r'\2 \1', s)

def func(m):

return m.group(1).title() + ' ' + m.group(2).title()

print p.subn(func, s)

### output ###

# ('say i, world hello!', 2)

# ('I Say, Hello World!', 2)

# -\*- coding: utf-8 -\*-

**import** re

#1. 匹配多个字符串

bt **=** 'bat|bet|bit'

m **=** re**.**search**(**bt**,** 'He bit bet'**)**

**if** m **is** **not** **None:**

**print** m**.**group**()**

m **=** re**.**findall**(**bt**,** 'He bit bet'**)**

**print** m

#2. 匹配单个字符(.)

patt314 **=** '3.14'

pi\_patt **=** '3\.14'

m **=** re**.**match**(**pi\_patt**,** '3.14'**)**

**if** m **is** **not** **None:** **print** m**.**group**()**

m **=** re**.**match**(**patt314**,** '3014'**)**

**if** m **is** **not** **None:** **print** m**.**group**()**

#3. 创建字符集合([])

m **=** re**.**match**(**'[cr][23][dp][o2]'**,** 'c3po'**)**

**if** m **is** **not** **None:** **print** m**.**group**()**

#4. 重复、特殊字符和子组

patt **=** '\w+@(\w+\.)?\w+\.com'

**print** re**.**match**(**patt**,** 'nobody@XXX.com'**).**group**()**

**print** re**.**match**(**patt**,** 'nobody@www.xxx.com'**).**group**()**

patt **=** '\w+@(\w+\.)\*\w+\.com'

**print** re**.**match**(**patt**,** 'nobody@www.xxx.yyy.com'**).**group**()**

m **=** re**.**match**(**'(\w{3})-(\d{3})'**,** 'abc-123'**)**

**print** m**.**group**()** #所有匹配的部分,和m.group(0)相同

**print** m**.**group**(**1**)** #匹配的子组1

**print** m**.**group**(**2**)** #匹配的子组2

**print** m**.**groups**()** #所有匹配子组 ('abc','123')

m **=** re**.**match**(**'(a(b))'**,**'ab'**)** #两个子组

**print** m**.**group**(),**m**.**group**(**1**),**m**.**group**(**2**),**m**.**groups**()**

#5. 从字符串的开头或结尾匹配及在单词边界上的匹配

re**.**search**(**'^The'**,** 'The end.'**)** #匹配

re**.**search**(**'^The'**,** 'end.The'**)** #不匹配

re**.**search**(**r'\bthe'**,**'bit thedog'**)** #匹配,单词左边界

#r'\bthe'中的r表示原始字符串(raw strings),即\b会被理解成一

#个正则表达式的一个特殊符号,而不是退格键

re**.**search**(**r'\bthe'**,** 'bit the dog'**)** #匹配

re**.**search**(**r'\bthe'**,**'bitthe dog'**)** #不匹配

re**.**search**(**r'\Bthe'**,** 'bitthe dog'**)** #匹配,单词右边界

re**.**search**(**r'\Bthe'**,** 'bit the dog'**)** #不匹配

#6. 用findall()找到每个出现的匹配部分

#结果: ['a','a','a']

**print** re**.**findall**(**'c(.)r'**,** 'carry the barcardi to the car'**)**

#结果: ['car', 'car', 'car']

**print** re**.**findall**(**'c.r'**,** 'carry the barcardi to the car'**)**

#7. 用sub()[和subn()]进行搜索和替换

**print** re**.**sub**(**'X'**,**'Mr. Smith'**,** 'attn: X\n\nDear X,\n'**)**

#结果: ('XbcdXf', 2)

**print** re**.**subn**(**'[ae]'**,** 'X'**,** 'abcdef'**)**

#8. 用split()分割(分割模式)

# re模块和正则表达式的方法split()与字符串的split()方法相似,

#前者根据正则表达式模式分割

# 字符串,后者根据固定的字符串分割.

#结果: ['123', '', '456', '789']

**print** "123\n\n456\n789"**.**split**(**'\n'**)**

#结果: ['123', '456', '789']

**print** re**.**split**(**'\n+'**,** "123\n\n456\n789"**)**

#9. 贪心匹配

# 正则表达式本身默认是贪心匹配的。即,如果正则表达式模式中

#使用到通配字,那它在按照从左到右

# 的顺序求值时,会尽量抓取满足模式的最长字符串.

testdata **=** 'gov::123143294538-6-8'

patt **=** '.+(\d+-\d+-\d+)'

**print** re**.**match**(**patt**,** testdata**).**group**(**1**)** #结果: 8-6-8

#结果: 123143294538-6-8

**print** re**.**search**(**'\d+-\d+-\d+'**,** testdata**).**group**()**

# 解决贪心匹配的一个办法是用"非贪婪"操作符"?",

#这个操作符可以用在\*,+,或?的后面.它的作用是要求

# 正则表达式引擎匹配的字符越少越好.

#结果: 123143294538-6-8

**print** re**.**match**(**'.+?(\d+-\d+-\d+)'**,** testdata**).**group**(**1**)**

# Django框架

## ****安装****

将下载后的Django-1.5.1.tar.gz解压到一个文件夹，打开cmd命令提示符，使用cd命令进入到解压后的文件夹中，再按照安装界面输入命令即可。

python setup.py install

linux系统安装需要使用root权限，可以用sudo命令，如下：

sudo python setup.py install

当然最简单的方法还是用easy\_install命令

sudo easy\_install django

以上步骤安装好以后再进行第二步。

第二步：将django-admin.py这个文件所在的目录加入到计算机的系统变量Path中去。

一般为：D:\Program Files\Python27\Scripts

第三步：验证Django是否安装成功

打开python IDLE

输入：

>>> import django

>>> django.VERSION

显示： (1, 5, 1, 'final', 0) 则安装成功。

## ****入门****

首先新建一个文件夹，如我新建的文件夹目录为：D:\djcode，打开cmd命令提示符窗口，用cd命令进入到djcode文件夹，使用下面的命令创建一个工程。

django-admin.py startproject firstsite

注意：我的python和django-admin都是加入过系统变量的可以这么写，如果没有加入的话还是使用python IDLE比较好。这样你就成功创建了一个名为“firstsite”的工程。可以看到其目录结构如下。

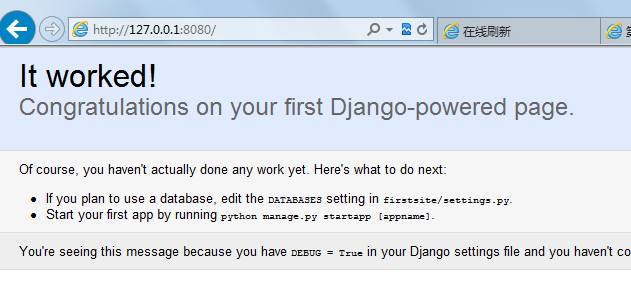
D:\DJCODE\FIRSTSITE  
│  manage.py  
│  
└─firstsite  
        settings.py  
        urls.py  
        wsgi.py  
        \_\_init\_\_.py

firstsite文件夹里面有manage.py文件和firstsite这个文件夹，firstsite这个文件夹里面又含有四个文件，读者可以自行打开看看，以后会慢慢讲解。

完成工程创建后就是启动web服务，进入到manage.py的同一级目录，使用下面的命令：

python manage.py runserver 0.0.0.0:8080

后面的8080如果不写的话默认是8000,。打开浏览器，输入：[http://127.0.0.1:8080](http://127.0.0.1:8080/)访问，出现下图则表示安装成功。加入0.0.0.0:8080是为了让任何人都可以访问到。



# Python中的进程与线程

## 进程和线程

进程和线程作为操作系统中的两个重要概念，即有相似之处，也有不同之处。本节首先介绍两者的概念，然后对Python的实现方式进行简要介绍。

与进程和线程相关的Python模块

|  |  |
| --- | --- |
| 模块名称 | 说明 |
| os/sys | 包含基本进程管理函数 |
| subprocess | Python基本库中多进程相关模块 |
| signal | Python基本库中信号相关模块 |
| threading | Python基本库中线程相关模块 |

## 进程的运行环境

在Python中，os模块提供了environ属性，用来记录当前进程的运行环境，这是一个字典型的数据结构，其中key为环境变量的变量名，value为环境变量的值。

#-\*-coding:utf-8-\*-

**import** os

# 获取PATH的环境变量

path **=** os**.**environ**.**get**(**'PATH'**)**

**print** path

# 生成一个新的环境变量

os**.**environ**[**'chuguangming'**]** **=** 'chuguangming'

# 获取所有的环境变量

**for** key **in** os**.**environ**.**keys**():**

**print** key**,** '\t'**,** os**.**environ**[**key**]**

## 创建进程

创建进程是系统管理的重要组成部分，Python语言提供了多种方式来创建进程，除了在os模块中有丰富的创建进程函数外，还有其他专门的模块如subprocess等来管理进程。在本节中主要对os模块中的进程创建函数进行说明。

在os模块中主要包含的system和exec家族函数，能够适应不周的创建进程需求，具体的将在下面进行介绍。同时，还有一些其他的创建进程方法，如os.popen等函数。

### system函数

system函数是用来创建进程的最快捷的方式，此函数在新进程中执行command字符串命令。如果返回值为0，表示命令执行成功，否则表示失败。现在，可以通过python来执行系统命令，如下代码：

#-\*-coding:utf-8-\*-

import os

print os.system('dir')

结果将输出这段代码所在目录的文件和文件夹。注意到程序最后还打钱出0，表示这次命令执行成功。

### exec家族函数

exec家住中有8个类似的函数（execl execle execlp execlpe execv execve execvp execvpe）。虽然用它们都可以创建进程，但是和system函数还是有些不同。system函数实际上是调用系统内置的命令行程序来执行系统命令，所以在命令结束之后会将控制权返回给Python进程。Python进程会在调用exec函数后终止。新生成的进程将会替换调用进程。这些函数没有返回值，如果发生错误，将会触发OSError异常。

下面是一个简单的例子：

#-\*-coding:utf-8-\*-

**import** os

notepad**=**'c:\\windows\\system32\\notepad.exe'

os**.**execl**(**notepad**,**"a.txt"**)**

## 终止进程

同创建进程的过程一样，终止进程的过程也是系统管理的重要组成部分。Python同样提供了多种不同的方式来终止进程。其实在前面就遇到过一种方法，那就是使用return语句。当Python脚本遇到最外层的return语句而退出的时候，这个进程就终止了。除此之外，还有两种方法终止进程：sys.exit和os.abort

### sys.exit函数

exit函数是一种温和的终止进程的方式，在程序退出之前会执行一些清理操作，同时将返回值给调用进程（一般是操作系统）。使用些返回值，系统可以判断程序是否正常退出或者运行出了异常。下面是一个程序的框架，用来读取用户给定的参数。

#-\*-coding:utf-8-\*-

**import** sys

**try:**

filename**=**sys**.**argv**[**1**]**

**print** filename

**except:**

**print** 'usage:'**,**sys**.**argv**[**0**],**"filename"

sys**.**exit**(**1**)**

### os.abort函数

和exit函数不同的是，abort函数是一种暴力的退出方式，将会直接给进程发送终止信息(SIGABORT信号)。在默认情况下，这将会终止进程，同时不会做相关的清理操作。需要注意的是，可以使用signal.signal()来为SIGABORT信号注册不同的信号处理函数，从而改变其默认行为。

一般地，在使用abort函数的地方都可以使用exit函数来替代，这样的终止程序方式更加优雅，但是在exit不能终止程序或者运行时间过长的情况下，可以尝试使用abort函数来解决。

## 使用subprocess模块管理进程

### 使用Popen类管理进程

演示用Popen类的方法，向sina发送ping报文，并获取相应的输出

#-\*-coding:utf-8-\*-

**import** subprocess

pingP**=**subprocess**.**Popen**(**args**=**'ping -n 4 www.sina.com.cn'**,**shell**=True)**

**print** pingP**.**pid #打印进程ID

**print** pingP**.**returncode #打印进程返回值

等待进程完成

#-\*-coding:utf-8-\*-

**import** subprocess

pingP**=**subprocess**.**Popen**(**args**=**'ping -n 4 www.sina.com.cn'**,**shell**=True)**

**print** pingP**.**wait**()**#等待进程完成

**print** pingP**.**pid #打印进程ID

**print** pingP**.**returncode #打印进程返回值

收集子进程的输出结果

#-\*-coding:utf-8-\*-

**import** subprocess

pingP **=** subprocess**.**Popen**(**args**=**'ping -n 4 www.sina.com.cn'**,**

shell**=True,** stdout**=**subprocess**.**PIPE**)**

**print** pingP**.**wait**()** # 等待进程完成

**print** pingP**.**stdout**.**read**()**#读取进程的输出信息

**print** pingP**.**pid # 打印进程ID

**print** pingP**.**returncode # 打印进程返回值

## 多线程概述

使用多线程处理技术，可以有效地实现程序并发，优化处理能力。同时，功能的更小划分，可以使代码的可重用性更好。

### 什么是多线程

多线程使得系统可以在单独的进程中执行并发任务。虽然进程也可以在独立的内存空间并发执行，但生成一个新进程必须为其分配单独的地址空间，并维护其代码段、堆栈段、数据段等，这种开销非常昂贵的。另外，进程间的通信实现也不方便。在程序功能日益复杂后，需要有更好的系统模型来满足要求。由此产生了线程。

线程是“轻量级”的，一个进程中的线程使用同样的地址空间，且共享许多资源。启动线程的时间远远小于启动进程的时间与空间，同时线程间的切换也要比进程间切换快得多。由于使用同样的地址空间，所以在线程之间的数据通信比较方便。一个进程下的线程之间可以直接使用彼此的数据。当然这种方便性也会带来一些问题，特别是同步问题。

多线程对于那些I/O受限的程序特别适用，其实多线程使用的一个重要目的，就是最大化CPU的资源利用。当某一线程在等待IO的时候，另外一个线程可以占用CPU资源。如最简单的GUI程序，一般需要有一个任务支持前台界面的交互，同时还有一个任务支持后台的处理。这时候，就适合采用线程模型，因为前台界面一般是在等待用户的输入操作，包括键盘输入和鼠标单击等。除此之外，多线程在网络领域和嵌入式领域应用也是比较多的。

### 线程的状态

在一个线程的生命周期内，会在不同的状态之间切换。在任何一个时刻，线程总是处于某种线程状态中，虽然不同的操作系统可以实现不同的线程模型，定义不同的线程状态，但是总的说来，一个线程模型中下面的状态是通用的。

1、就绪状态：线程已经获得了除CPU外的其他资源，正在参与调度，等待被执行。当被调度选中后，将立即执行。

2、休眠状态：暂时不参与调度，等待特定事件发生，比如IO事件。

3、运行状态：占用CPU资源，正在系统中运行。

4、中止状态：线程已经运行结束，等待系统回收线程资源。

Python使用全局解释器锁(GIL)来保证在解释器中仅仅只有一个线程，并在各个线程之间切换。当GIL可用的时候，处于就绪状态的线程在获取GIL后就可以运行了。线程将在指定的间隔时间内运行。当时间到期后，正在执行的线程将重新进入就绪状态并排队。GIL重新可用并且为就绪状态的线程获取。当然，特定的事件也有可能导致中断正在运行的线程。

### Python中的线程支持

现在，Python语言中已经为各种平台提供了多线程处理能力，包括Windows/Linux等系统平台。在具体的库上，提供了两种不同的方式。一种是低级的线程处理模块thread，仅仅提供了一个最小的线程处理功能集，在实隙地的代码中最好是不要直接使用该模块的。另外一种是高级的线程处理模块threading，现在大部分应用的线程实现都是基于此模块的。threading模块是基于thread模块的，部分实现思想也来自于java的threads类。

多线程设计的最大问题是如何协调多个线程，因此，在threading模块中，提供了多种数据同步的方法。为了能够更好的实现线程同步，Python中提供了Queue模块，用来同步线程。在Queue模块中，有一个同步的FIFO队列类型，特别适合线程之间的数据通信和同步。

由于大部分程序不需要有多线程处理的能力，所以在Python启动的时候并不支持多线程。也就是说，Python中支持多线程所需要的各种数据结构特别是GIL还没有创建。当Python虚拟机启动的时候，多线程处理并没有打开，而仅仅只支持单线程。这样做的好处是使得系统处理更加高效。只有当程序中使用了如

thread.start\_new\_thread等方法的时候，Python才知道需要有多线程处理的支持。这时，Python虚拟机才会自动创建多线程创建多线程处理所需要的数据结构和GIL。

## 生成和终止线程

生成和终止线程是线程管理的关键步骤。在前面说过，只有生成了线程后，Python虚拟机才会开启对多线程处理的能力。由于Python对线程的支持即有低级方法，也有高级方法，所以生成和终止线程的方式也是多样的。

### 使用thread模块

thread模块作为低级模块，虽然不被推荐直接使用，但是因为其用法非常简洁，在某些简单场合也是可以试试的。其中的的核心是start\_new\_thread方法，可以用来生成线程。此方法接受一个函数对象作为参数，同时还有此函数的参数列表，另外还有一个可选的字典参数。此方法将会返回生成新线程的标识符。

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** thread

**import** time**,** thread

**def** timer**():**

**print(**'hello'**)**

**def** test**():**

**for** i **in** range**(**0**,** 3**):**

thread**.**start\_new\_thread**(**timer**,** **())**

**if** \_\_name\_\_**==**'\_\_main\_\_'**:**

test**()**

time**.**sleep**(**10**)**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** time**,** thread

**def** timer**(**name**=None,** group**=None):**

**print(**'name: ' **+** name **+** ', group: ' **+** group**)**

**def** test**():**

**for** i **in** range**(**0**,** 3**):**

thread**.**start\_new\_thread**(**timer**,** **(**'thread' **+** str**(**i**),** 'group' **+** str**(**i**)))**

**if** \_\_name\_\_**==**'\_\_main\_\_'**:**

test**()**

time**.**sleep**(**10**)**

### thread的锁机制

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** time**,** thread

count **=** 0

**def** test**():**

**global** count

**for** i **in** range**(**0**,** 10000**):**

count **+=** 1

**for** i **in** range**(**0**,** 10**):**

thread**.**start\_new\_thread**(**test**,** **())**

time**.**sleep**(**5**)**

**print** count

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** time**,** thread

count **=** 0

lock **=** thread**.**allocate\_lock**()**

**def** test**():**

**global** count**,** lock

lock**.**acquire**()**

**for** i **in** range**(**0**,** 10**):**

count **+=** 1

lock**.**release**()**

**for** i **in** range**(**0**,** 10**):**

thread**.**start\_new\_thread**(**test**,** **())**

time**.**sleep**(**5**)**

**print** count

## 使用threading.Thread类

使用threading模块来创建线程是很方便的。简单地说，只有继承threading.Thread，然后在\_\_init\_\_方法中调用threading.Thread类中的\_\_init\_\_方法,重写类的run方法就可以了。

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

**class** **ThreadSkeleton(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

**def** run**(**self**):**

**pass**

thread**=**ThreadSkeleton**()**

thread**.**start**()**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

**import** time

**class** **ThreadSkeleton(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**,**index**,**create\_time**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

self**.**index**=**index

self**.**create\_time**=**create\_time

#具体的线程运行代码

**def** run**(**self**):**

time**.**sleep**(**1**)**#休眠1秒钟

**print** "Thread %d %f \n"**%(**self**.**index**,**time**.**time**()-**self**.**create\_time**)**

**for** index **in** range**(**5**):**

thread**=**ThreadSkeleton**(**index**,**time**.**time**())**

thread**.**start**()**

**print** "Main thread exit..."

### join等待线程完成

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

**import** time

**class** **ThreadSkeleton(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**,**index**,**create\_time**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

self**.**index**=**index

self**.**create\_time**=**create\_time

#具体的线程运行代码

**def** run**(**self**):**

time**.**sleep**(**1**)**#休眠1秒钟

**print** "Thread %d %f \n"**%(**self**.**index**,**time**.**time**()-**self**.**create\_time**)**

**for** index **in** range**(**5**):**

thread**=**ThreadSkeleton**(**index**,**time**.**time**())**

thread**.**start**()**

#等待线程完成

thread**.**join**()**

**print** "Main thread exit..."

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

**import** time

**class** **ThreadSkeleton(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**,**index**,**create\_time**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

self**.**index**=**index

self**.**create\_time**=**create\_time

#具体的线程运行代码

**def** run**(**self**):**

time**.**sleep**(**1**)**#休眠1秒钟

**print** "Thread %d %f \n"**%(**self**.**index**,**time**.**time**()-**self**.**create\_time**)**

threads**=[]**

**for** index **in** range**(**5**):**

thread**=**ThreadSkeleton**(**index**,**time**.**time**())**

thread**.**start**()**

threads**.**append**(**thread**)**

**for** thread **in** threads**:**

thread**.**join**()**

**print** "Main thread exit..."

### 线程中的局部变量

有时间需要在每个线程中使用各自独立的变量，一个显而易见的方法就是每个线程都使用私有变量。为了方便Python中提供了一种简单的机制来解决这个问题。这就是thread.local。其使用方法也很简单，其成员变量在每个线程中都是不同的。

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

**import** time

**import** random

**class** **ThreadLocal(**threading**.**Thread**):**

**def** \_\_init\_\_**(**self**):**

threading**.**Thread**.**\_\_init\_\_**(**self**)**

#生成local数据对象

self**.**local**=**threading**.**local**()**

**def** run**(**self**):**

#生成随机休眠时间

time**.**sleep**(**random**.**random**())**

self**.**local**.**number**=[]**

**for** i **in** range**(**10**):**

self**.**local**.**number**.**append**(**random**.**choice**(**range**(**10**)))**

**print** threading**.**currentThread**(),**self**.**local**.**number

threads**=[]**

**for** i **in** range**(**5**):**

t**=**ThreadLocal**()**

t**.**start**()**

threads**.**append**(**t**)**

**for** i **in** range**(**5**):**

threads**[**i**].**join**()**

## 线程之间的同步

由于同一进程中的所有线程都是共享数据的，如果对线程中数据的并发访问不加以限制，结果将不可预测。在严重的情况下，还会产生死锁。为了解决这个问题，需要允许线程独占性的访问共享数据，这就是线程同步。

### 临界资源和临界区

临界资源是指一次只允许一个线程访问的资源,包括如打印机一类的硬件资源和互斥变量一类的软件资源。对临界资源的共享只能够采用互斥的方式。也就是说，在一个线程访问的时候，其他线程必须等待。此时，线程之间不能交替使用该资源，否则就会导致执行结果的不可预测期和不一致性。一般地，线程中访问临界资源的代码部分被称为临界区。

简单地说，访问临界区的代码不能同时执行。在线程进入临界区之前，首先要检查是否已经有线程在访问临界资源。如果临界资源空闲，才可以进入临界区执行，并设置访问标志，使得其他线程不能再进入临界区。如果临界资源忙，该线程需要等待，直到临界资源被释放。

有一个全局的计数num，每个线程获取这个全局的计数，根据num进行一些处理，然后将num加1。很容易写出这样的代码：

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**import** threading

# encoding: UTF-8

**import** threading

**import** time

**class** **MyThread(**threading**.**Thread**):**

**def** run**(**self**):**

**global** num

time**.**sleep**(**1**)**

num **=** num**+**1

msg **=** self**.**name**+**' set num to '**+**str**(**num**)**

**print** msg**+**"\n"

num **=** 0

**def** test**():**

**for** i **in** range**(**5**):**

t **=** MyThread**()**

t**.**start**()**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

test**()**

问题产生的原因就是没有控制多个线程对同一资源的访问，对数据造成破坏，使得线程运行的结果不可预期。这种现象称为“线程不安全”。

上面的例子引出了多线程编程的最常见问题：数据共享。当多个线程都修改某一个共享数据的时候，需要进行同步控制。

线程同步能够保证多个线程安全访问竞争资源，最简单的同步机制是引入互斥锁。互斥锁为资源引入一个状态：锁定/非锁定。某个线程要更改共享数据时，先将其锁定，此时资源的状态为“锁定”，其他线程不能更改；直到该线程释放资源，将资源的状态变成“非锁定”，其他的线程才能再次锁定该资源。互斥锁保证了每次只有一个线程进行写入操作，从而保证了多线程情况下数据的正确性。

threading模块中定义了Lock类，可以方便的处理锁定：

**import** threading

**import** time

**class** **MyThread(**threading**.**Thread**):**

**def** run**(**self**):**

**global** num

time**.**sleep**(**1**)**

**if** mutex**.**acquire**(**1**):**

num **=** num**+**1

msg **=** self**.**name**+**' set num to '**+**str**(**num**)**

**print** msg

mutex**.**release**()**

num **=** 0

mutex **=** threading**.**Lock**()**

**def** test**():**

**for** i **in** range**(**5**):**

t **=** MyThread**()**

t**.**start**()**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

test**()**

# Pyexe

有的时候大部分人对于python这种脚本化的运行方式非常的不爽，其实解决方式很多，今天给出一种方法。PyInstaller. PyInstaller可以将Python程序打包成Windows（当然也包括Linux, Mac OS X, Solaris and AIX）下可执行的EXE文件，目前支持python2.2-2.7版本，点击这里下载。

使用PyInstaller需要安装PyWin32，可到这里下载相应的版本。（从pywin32的下载量看，还是Python2.7使用更广泛）

PyInstaller下载地址：http://www.pyinstaller.org/

PyWin32下载地址：http://sourceforge.net/projects/pywin32/files/pywin32/Build%20217/



1、下载对应已安装的Python版本的PyInstaller版本，解压到任意目录，按照提供的manual文档进行即可

2、假设要转换的Python代码为

C:\PythonProject\testurulib\urllib2test1.py

3、进入PyInstaller安装根目录，执行以下命令。

python PyInstaller.py C:\PythonProject\testurulib\urllib2test1.py

4、结果



常用参数：

-F　　　　制作独立的可执行程序

-D　　　　制作出的档案存放在同一个文件夹下（默认值）

-K　　　　包含TCL/TK（对于使用了TK的，最好加上这个选项，否则在未安装TK的电脑上无法运行）

-w　　　 制作窗口程序

-c　　　　制作命令行程序（默认）

-X　　　　制作使用UPX压缩过的可执行程序

UPX下载地址：http://upx.sourceforge.net/download/upx308w.zip

-o DIR　　指定输出SPEC文件路径（这也决定了最后输出的exe文件路径）

--icon=[ICO文件路径] 指定程序图标

-v [指定文件] 指定程序版本信息

-n [指定程序名] 指定程序名称

具体参数见：安装文档，非常的丰富

# Sina SDK的使用

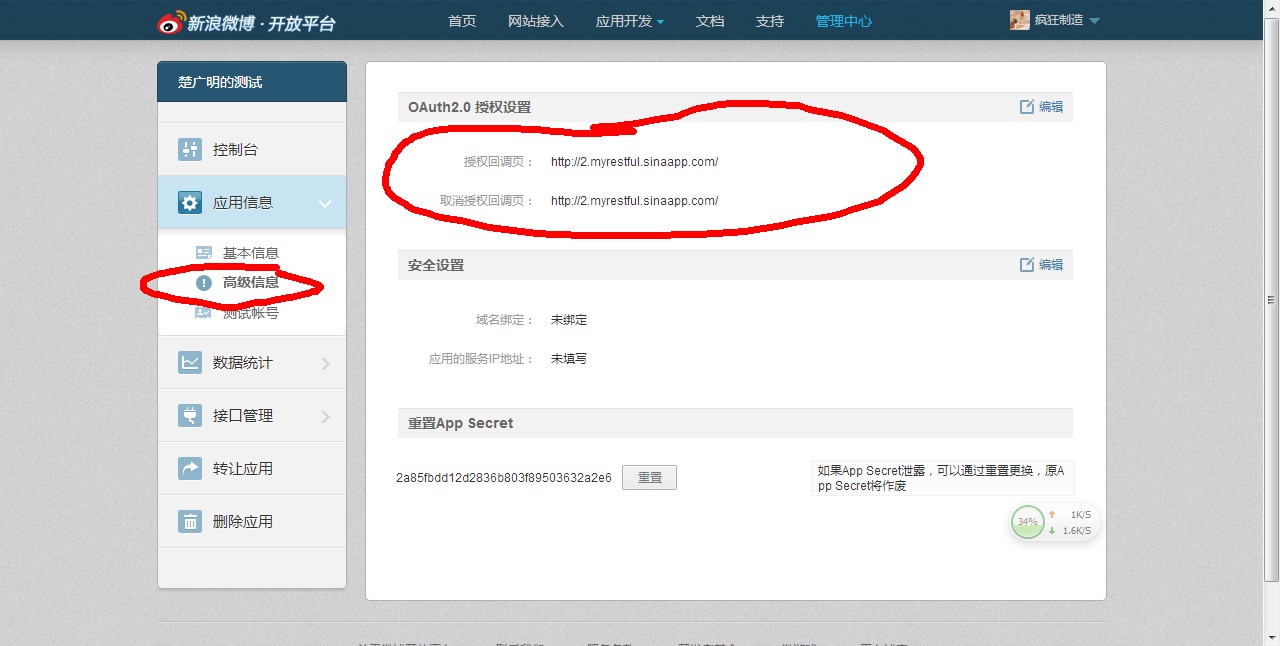
## 首先申请SDK开发者

1、首先访问<http://open.weibo.com/>申请开发者





在申请的时候其中很简单，大家可以参考SDK的帮助，唯一要注意的是，回调地址一定要真实，要不以后会有麻烦的。



## 下载Python SDK

去下载SDK，地址：<http://michaelliao.github.io/sinaweibopy/>

## 一个简单的演示程序

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**from** weibo **import** APIClient

**import** sys

**import** webbrowser

APP\_KEY **=** '1705534647' # 填写申请的APP\_KEY

APP\_SECRET **=** '2a85fbdd12d2836b803f89503632a2e6' # 填写申请的APP\_SECRET

CALLBACK\_URL **=** 'http://2.myrestful.sinaapp.com/'

''' posttext.py 调用weibo SDK 命令发微博 '''

# 获取微博授权，手动操作

**def** getclient**():**

client **=** APIClient**(**app\_key**=**APP\_KEY**,** app\_secret**=**APP\_SECRET**,** redirect\_uri**=**CALLBACK\_URL**)**

url **=** client**.**get\_authorize\_url**()**

**print** url # 浏览器打开该url，取得code='xxx'类似的code输入

# 将地址保存在文件中，可以手工的打开这个文件，在地址栏中显示

# 之后转向到一个新的url

#

#https://api.weibo.com/oauth2/default.html?code=96296f963705cbca77b673ac7fbdf9d0

# 其中的code=后面就是code

f **=** file**(**"url.txt"**,**"w+"**)**

f**.**writelines**(**url**)**

f**.**close**()**

#sinaauthurl = urllib.urlopen(url)

#print sinaauthurl.read().decode('utf-8')

webbrowser**.**open\_new\_tab**(**url**)**

code **=** raw\_input**(**'Enter code >'**)**

r **=** client**.**request\_access\_token**(**code**)**

access\_token **=** r**.**access\_token

expires\_in **=** r**.**expires\_in

client**.**set\_access\_token**(**access\_token**,** expires\_in**)**

**return** client

# 发微博

**def** posttext**(**client**):**

text **=** raw\_input**(**'Enter text to post >'**)**

utext **=** unicode**(**text**,** "UTF-8"**)**

client**.**statuses**.**update**.**post**(**status**=**utext**)**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

client **=** getclient**()**

posttext**(**client**)**

## 一个进阶的演示

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

**from** weibo **import** APIClient

**import** sys

**import** webbrowser

APP\_KEY **=** '1705534647' # 填写申请的APP\_KEY

APP\_SECRET **=** '2a85fbdd12d2836b803f89503632a2e6' # 填写申请的APP\_SECRET

CALLBACK\_URL **=** 'http://2.myrestful.sinaapp.com/'

''' posttext.py 调用weibo SDK 命令发微博 '''

# 获取微博授权，手动操作

**def** getclient**():**

client **=** APIClient**(**app\_key**=**APP\_KEY**,** app\_secret**=**APP\_SECRET**,** redirect\_uri**=**CALLBACK\_URL**)**

url **=** client**.**get\_authorize\_url**()**

**print** url # 浏览器打开该url，取得code='xxx'类似的code输入

# 将地址保存在文件中，可以手工的打开这个文件，在地址栏中显示

# 之后转向到一个新的url

# https://api.weibo.com/oauth2/default.html?code=96296f963705cbca77b673ac7fbdf9d0

# 其中的code=后面就是code

f **=** file**(**"url.txt"**,**"w+"**)**

f**.**writelines**(**url**)**

f**.**close**()**

# sinaauthurl = urllib.urlopen(url)

# print sinaauthurl.read().decode('utf-8')

webbrowser**.**open\_new\_tab**(**url**)**

code **=** raw\_input**(**'Enter code >'**)**

r **=** client**.**request\_access\_token**(**code**)**

access\_token **=** r**.**access\_token

expires\_in **=** r**.**expires\_in

client**.**set\_access\_token**(**access\_token**,** expires\_in**)**

**return** client

# 发微博

**def** posttext**(**client**):**

text **=** raw\_input**(**'Enter text to post >'**)**

utext **=** unicode**(**text**,** "UTF-8"**)**

client**.**statuses**.**update**.**post**(**status**=**utext**)**

#绝大部分都需要获取用户信息

**def** getmyinfo**(**client**):**

r **=** client**.**users**.**show**.**get**(**screen\_name**=**'疯狂制造'**)**

**return** r

**def** getmyfanscount**(**client**,**id**):**

r**=**client**.**users**.**counts**.**get**(**uids**=**id**)**

**return** r

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

client **=** getclient**()**

allfans**=** getmyinfo**(**client**)**

#得到用户ID

**print** allfans**.**id

**print** allfans**.**description

#得到用户有多少fans

**print** getmyfanscount**(**client**,**allfans**.**id**)**

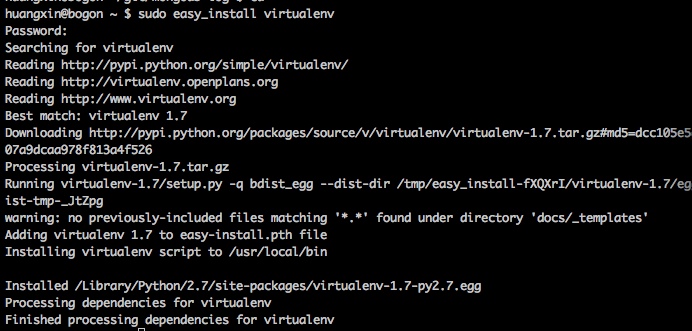
# posttext(client)

# virtualenv

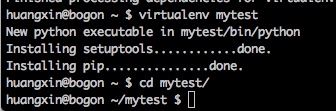
Python的背后有着庞大的开源社区支持，但是有一个缺点就是每个包的质量都参差不齐，如果我们在工作服务器上去测试安装每个包，就会造成整个的服务器形成庞大复杂的第三方包依赖。virtualenv就是为了解决这个问题而生，他可以在我们的目录上生成若干个独立的虚拟环境，这些虚拟环境和正式环境一样。

virtualenv简介 VirtualEnv用于在一台机器上创建多个独立的python运行环境，类似于一个个沙盒环境。 作用： 隔离项目之间的第三方库依赖，如A项目依赖xxx1.1，B项目依赖xxx1.2。 为部署应用提供方便，把开发环境的虚拟环境打包到生产环境即可,不需要在服务器上再折腾一翻。 尝试新的第三方库，而不用担心污染系统环境

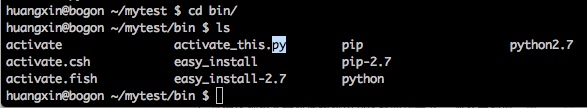
## 安装virtualenv



然后我们就可以创建一个虚拟的环境了：



mytest就是我们所创建的虚拟环境目录了，我们可以发现，该目录中已经安装了和我们本机相同的python版本：



我们接下来需要激活这个虚拟环境：

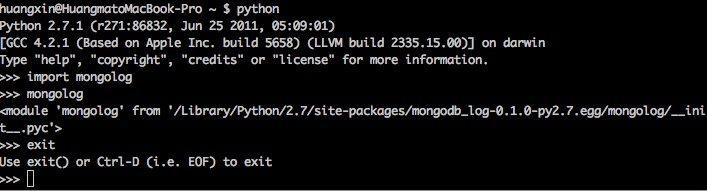
http://images.cnblogs.com/cnblogs_com/kym/4.jpeg

我们就可以看到在命令行的前面已经加上了“mytest”，这个时候我们看看python所指向的路径：

http://images.cnblogs.com/cnblogs_com/kym/5.jpeg

OK ，这个时候我们就可以在这个虚拟环境中安装我们所需要的包了：





## 扩展virtualenv的工具

virtualenv --clear mysite #清除虚拟环境mysite

$ /$path/python/bin/virtualenv.py --no-site-packages vm01

//创建虚拟环境vm01，增加可选参数--no-site-packages作用：在建立虚拟环境时，会依赖系统环境中的site-packages，如果想完全不依賴系統的packages，可以加上此参数來建立虚拟环境。

(vm01)$ source ./vm01/bin/activate

//启动虚拟环境

(vm01)$ deactivate

//退出虚拟环境

$ easy\_install virtualenv

$ easy\_install virtualenvwrapper

## python多版本管理利器-pythonbrew

Pythonbrew 可以在多个 Python之间迅速切换，也可以在指定的 Python 版本下测试python程序，所以，当你的电脑里装了多个python版本，并需要经常切换测试时，pythonbrew就非常适合你。

安装配置pythonbrew

alexzhou@alexzhou:~/python\_workspace$ sudo easy\_install pythonbrew

alexzhou@alexzhou:~/python\_workspace$ pythonbrew\_install

添加：

[[ -s "$HOME/.pythonbrew/etc/bashrc" ]] && source "$HOME/.pythonbrew/etc/bashrc"

到~/.bashrc最后。然后执行：source ~/.bashrc

查看系统可以安装的python版本

alexzhou@alexzhou:~/python\_workspace$ pythonbrew list --know

# Pythons

Python-1.5.2

Python-1.6.1

Python-2.0.1

Python-2.1.3

Python-2.2.3

Python-2.3.7

Python-2.4.6

Python-2.5.6

Python-2.6.7

Python-2.7.3

Python-3.0.1

Python-3.1.4

Python-3.2.3

安装最新的版本和2.6.7版本，ps：需要使用curl安装python，所以需要先安装curl

alexzhou@alexzhou:~/python\_workspace$ pythonbrew install 3.2.3

pythonbrew required curl. curl was not found in your path.

alexzhou@alexzhou:~/python\_workspace$ sudo apt-get install curl

alexzhou@alexzhou:~/python\_workspace$ pythonbrew install 3.2.3

Downloading Python-3.2.3.tgz as /home/alexzhou/.pythonbrew/dists/Python-3.2.3.tgz

......

alexzhou@alexzhou:~/python\_workspace$ pythonbrew install 2.6.7

清理安装后的版本的源码和安装包

alexzhou@alexzhou:~/python\_workspace$ pythonbrew cleanup

查看帮助

alexzhou@alexzhou:~/python\_workspace$ pythonbrew --help

选择一个python版本使用，ps：只在当前终端有效

alexzhou@alexzhou:~/python\_workspace$ pythonbrew use 2.6.7

Using `Python-2.6.7`

查看当前pythonbrew下的python版本有哪些（后面有\*号表示正在使用）

alexzhou@alexzhou:~/python\_workspace$ pythonbrew list

# pythonbrew pythons

Python-2.6.7 (\*)

Python-3.2.3

选择python2.6.7版本作为系统默认版本使用，会把该版本的路径添加到PATH中

alexzhou@alexzhou:~/python\_workspace$ pythonbrew switch 2.6.7

取消pythonbrew选择的版本

alexzhou@alexzhou:~/python\_workspace$ pythonbrew off

使用virtualenv的功能，创建虚拟环境  
首先选择一个python版本

alexzhou@alexzhou:~/python\_workspace$ pythonbrew switch 2.6.7

创建虚拟环境

alexzhou@alexzhou:~/python\_workspace$ pythonbrew venv create test\_env

虚拟环境列表

alexzhou@alexzhou:~/python\_workspace$ pythonbrew venv list

# virtualenv for Python-2.6.7 (found in /home/alexzhou/.pythonbrew/venvs/Python-2.6.7)

test\_env

启用虚拟环境

alexzhou@alexzhou:~/python\_workspace$ pythonbrew venv use test\_env

# Using `test\_env` environment (found in /home/alexzhou/.pythonbrew/venvs/Python-2.6.7)

# To leave an environment, simply run `deactivate`

(test\_env)alexzhou@alexzhou:~/python\_workspace$

退出虚拟环境

(test\_env)alexzhou@alexzhou:~/python\_workspace$ deactivate

# 附录-Ubuntu开发环境之安装

内容来之于<http://teliute.org>，非常好的新手教程网站

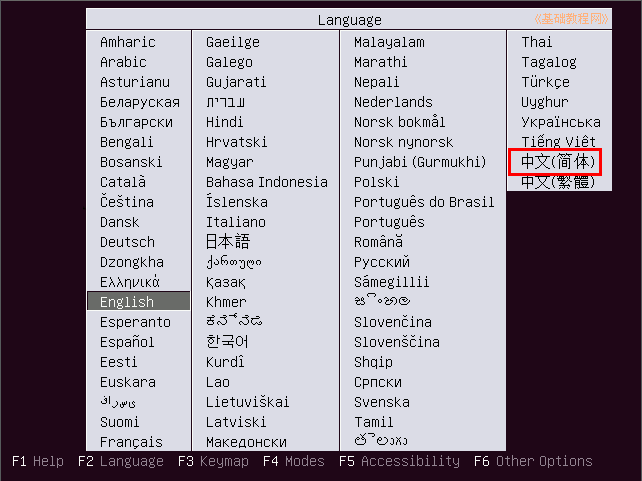
## 进入 live cd 桌面

　1）设置好启动后，断开网络，然后重启动计算机，可以用硬盘启动，也可以刻成光盘启动，镜像的下载地址：

　进入后找蓝色链接点击下载，如 ubuntu-12.04-desktop-i386.iso，64位CPU可以下载amd64的版本：

<http://mirrors.sohu.com/ubuntu-releases/12.04/>

　2）启动后稍等，系统自动运行，在下边出来两个图标时a2tb.png，可以按一下 ESC键呼出菜单项，再按右方向键选择“中文(简体)”，按回车键；



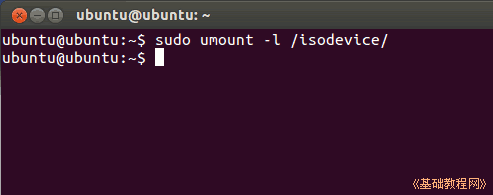
　3）等一会就进入一个桌面，这就是试用的 live cd 桌面，桌面左上边有有两个图标，右上角是“关机”按钮；



　4）对于硬盘安装，点一下左上角的圆圈按钮，稍等在旁边出来的文本框中输入字母 ter 然后点击下边出来的终端图标；



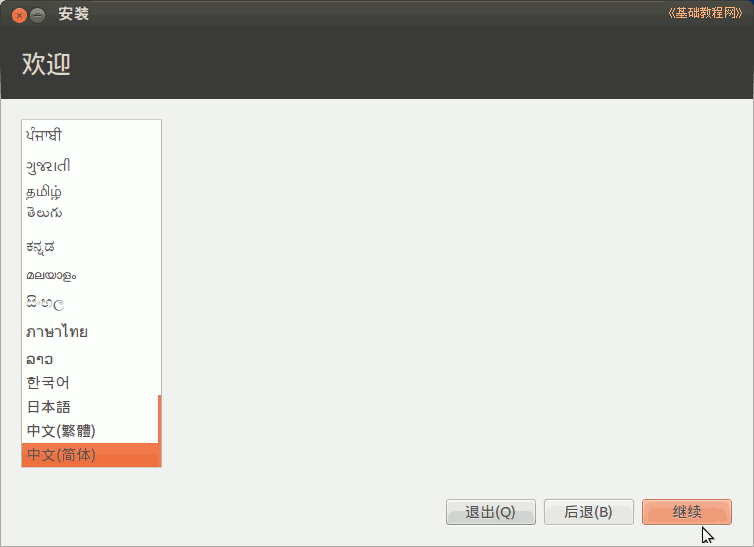
　5）输入命令 sudo umount -l /isodevice 然后按一下回车键，没什么提示就是成功了，关闭终端；



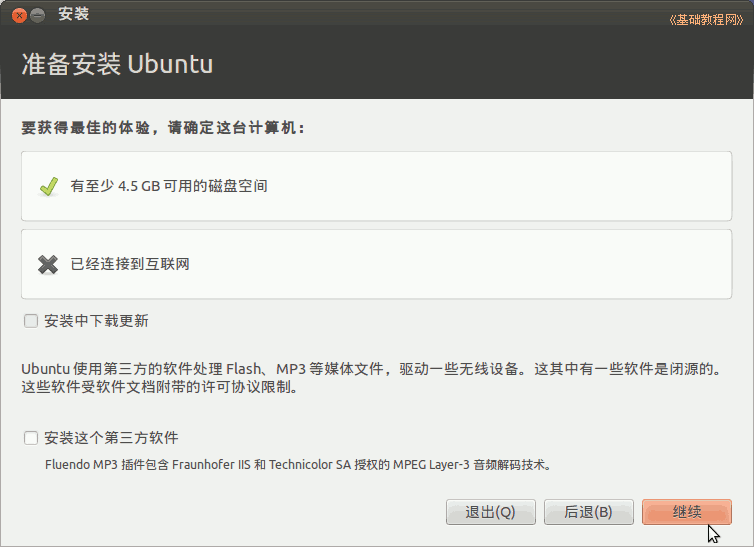
## 安装系统

　1）双击桌面“安装Ubuntu12.04 LTS”图标，稍等出来一个“欢迎”面板，左侧应该选中了“中文(简体)”，

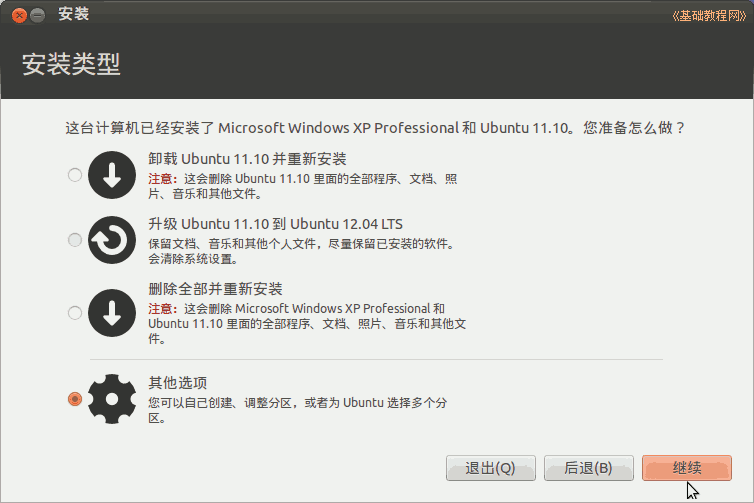
　如果不是就在左边选中它，然后点右下角“继续”按钮；



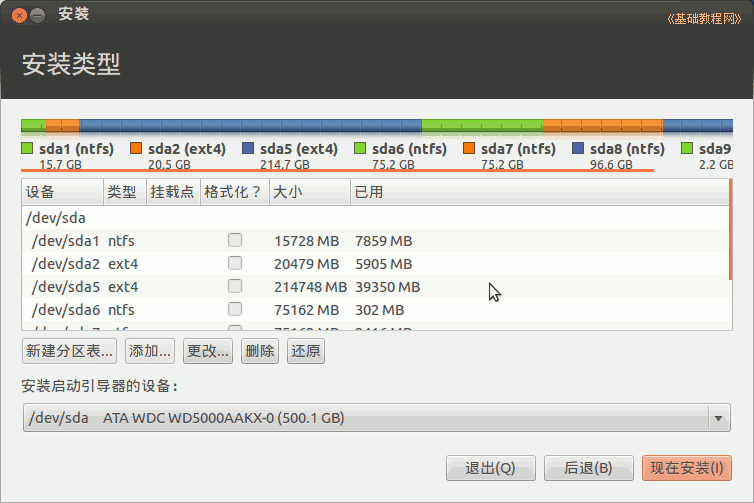
　2）第2步是检查准备情况，要求磁盘空间足够，不要连接网络，一般不勾选更新和第三方软件，直接点“继续”按钮；



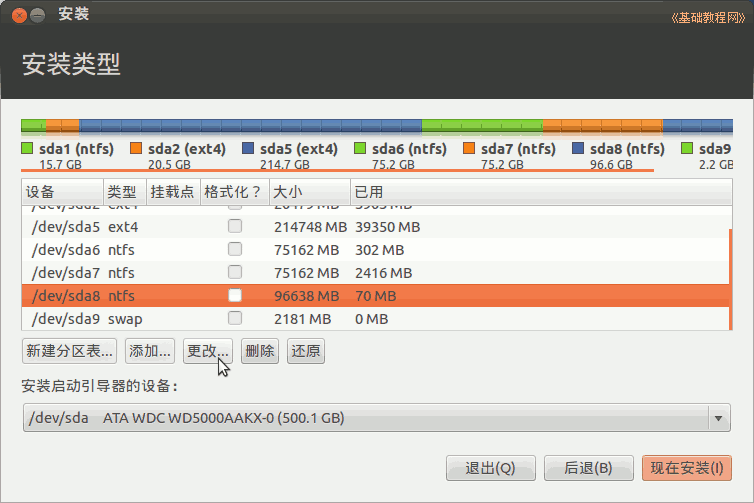
　3）接下来第3步是询问安装到哪个分区，选择最下边的“其他选项”，点“继续”按钮；



　4）接下来出来磁盘分区情况，如果要新建分区和转换分区可以参考前面的第7、13、15、16、19课，这儿是安装到一个NTFS分区上；

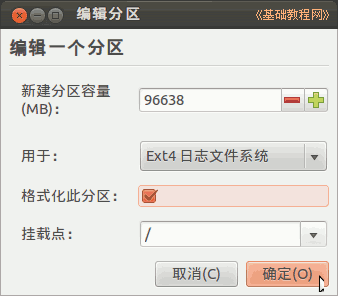


　5）点击选中计划要安装的分区，可以根据分区类型和大小来确定，然后点下边的“更改”按钮；

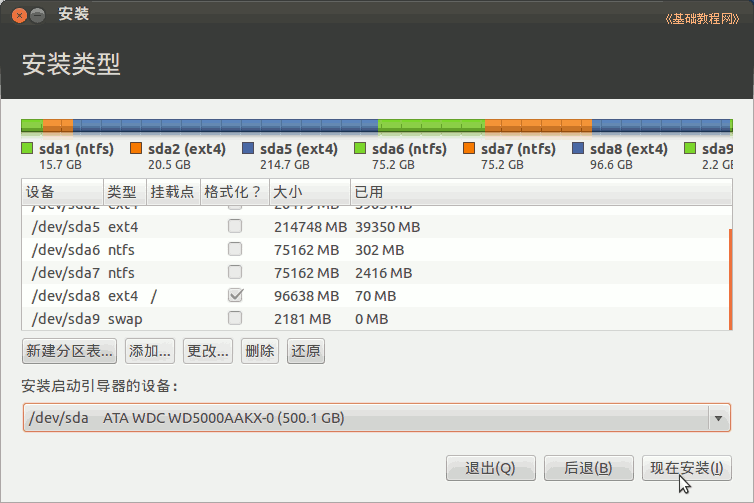


　6）在出来的对话框中，设定用于分区的格式Ext4，打勾“格式化”，在“挂载点”右边点一下，选 “/”，点“确定”

　注意，格式化会删除这个分区上的所有文件，请提前备份重要数据；

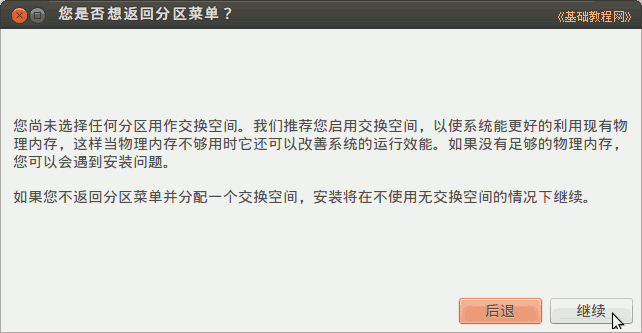


　7）回到分区面板，检查一下分区编辑好了，点“现在安装”，如果还有 /home 分区，按原来的设，一般不打勾“格式化”，只需提前清理里面的配置文件；

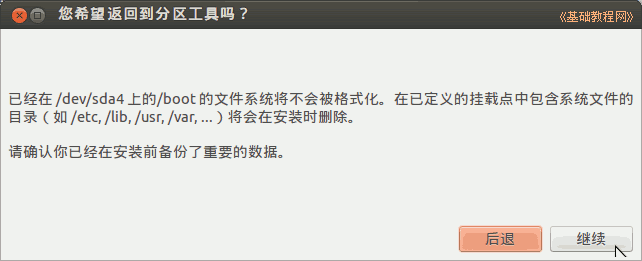


　8）在下边是安装grub引导器的选项，可以选择安装到/分区，然后用Windows来引导Ubuntu，参阅：[用 EasyBCD 在 Win 7/8 硬盘安装 Ubuntu](http://teliute.org/linux/Ubsetup/jichu3/jichu3.html)

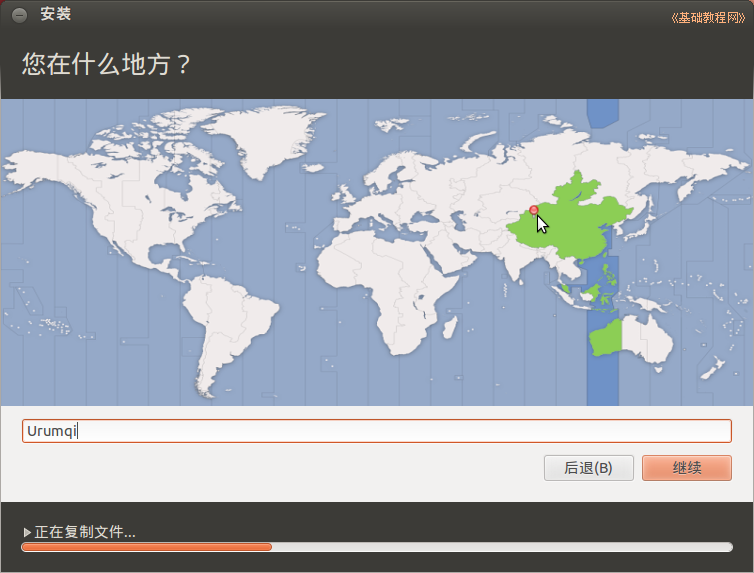
　如果提示没有交换空间，点“继续”，这个是用于休眠的，大小跟内存相同的分区，一般不用它；



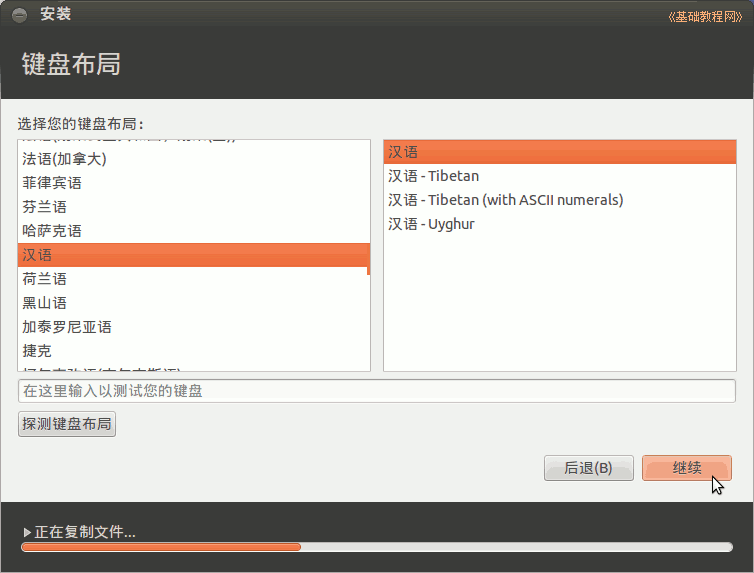
　提示即将删除和格式化时，确保重要数据已经备份，点“继续”；



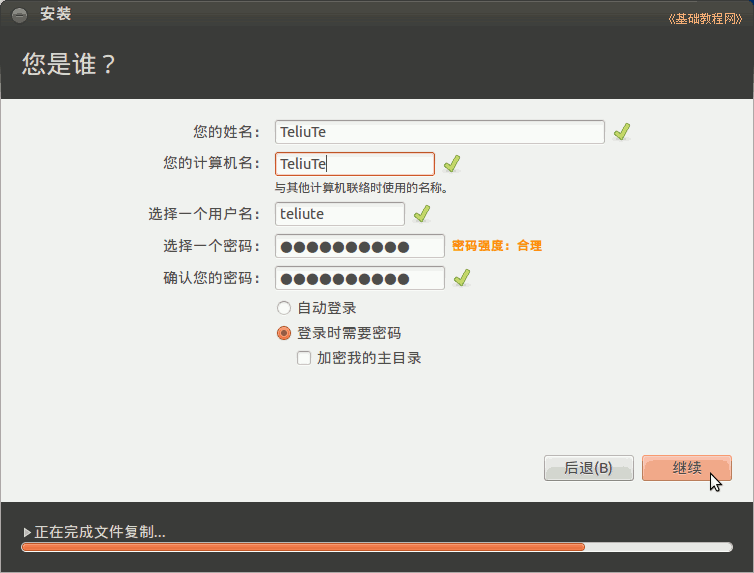
　9）然后出来询问地区，点“继续”即可，或者在地图上找找自己所在的地区，这时下边已经开始安装了；



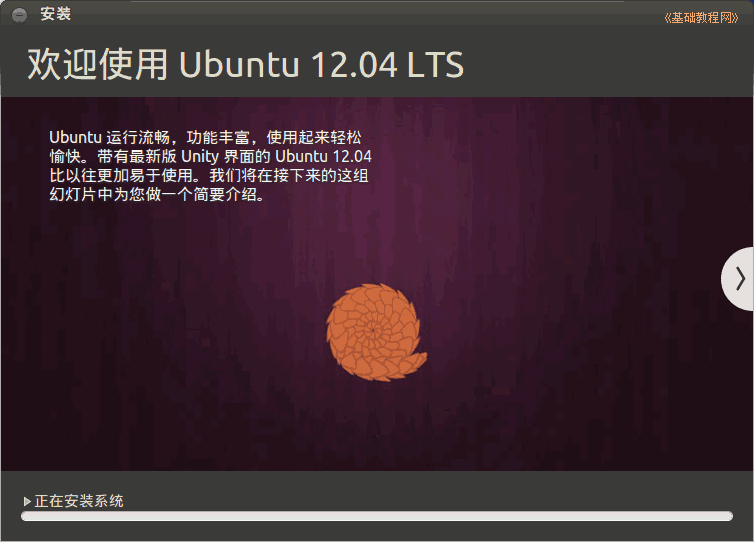
　10）接下来是键盘布局，一般是“USA”，直接点“前进”继续；



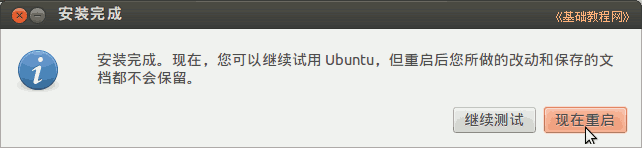
　11）接下来是设定自己的用户名(小写字母)和密码等，从上到下依次输入即可，然后点“继续”；



　12）然后接着继续安装过程，可以看一下系统的介绍；



　13）耐心等待完成，然后出来一个对话框，点“现在重启”完成安装，按电源键也可以呼出关机对话框；



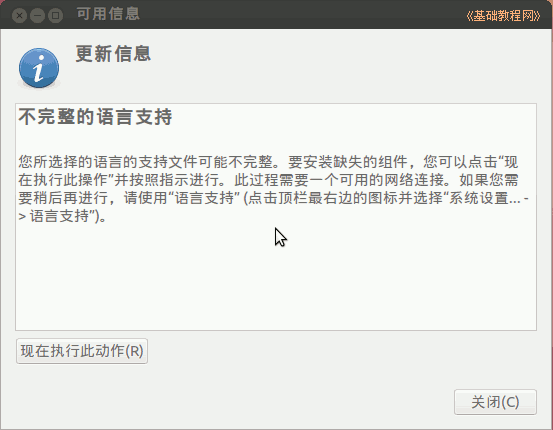
　14）如果是光盘安装，稍等提示取出光盘，然后按回车键，重新启动计算机，安装完成；

## 连网换源

　１）重新启动后，停在一个登录界面，点击自己的用户名，输入密码后按回车，进入系统；



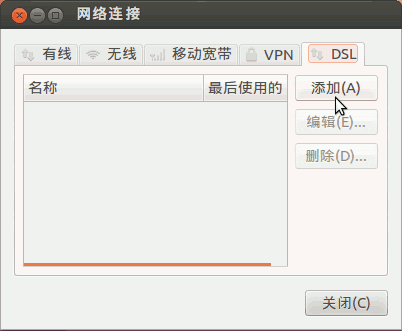
　2）进入桌面后会出来一个“不完整语言支持”的提示对话框，先不关闭拖到一边后面要用；



　3）先连网，找到屏幕右上角键盘旁边的一个网络图标，瞄准点左键，选“编辑连接”；

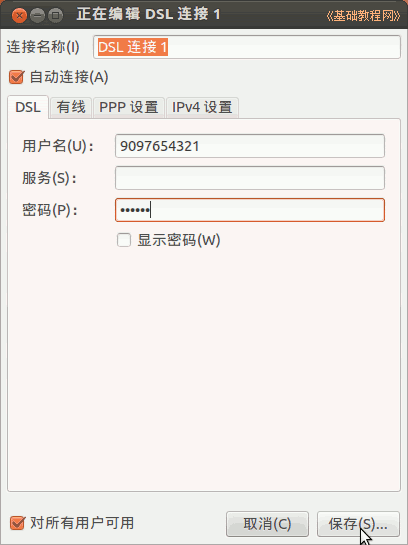


　4）对于ADSL拨号宽带连接上网，在出来的对话框右边，点 DSL 标签，然后点添加；



　5）在出来的对话框里，先打勾上边的 “自动连接”，然后在用户名里输入ADSL宽带用户名，

　下面的密码里，输入宽带密码，然后点右下角的“应用”按钮，在出来的密码框中输入密码；



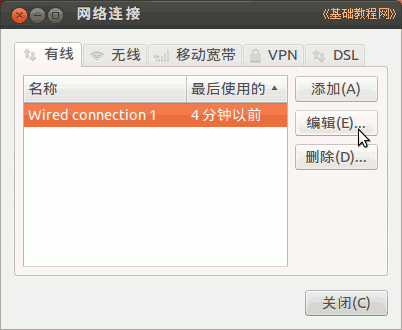
　关闭对话框以后，稍等一下会提示连网成功，



　也可以重新启动计算机，命令行使用 sudo pppoeconf 命令也可以；

　(下面6-7是局域网的，宽带用户不要设置)

　6）如果是局域网上网，在出来的对话框里，选中里面的“Wired connection1”，点右边的“编辑”按钮；



　7）在出来的对话框里，选择 IPV4 设置标签，在下面的方法里选“手动”，再点下边的“添加”按钮，

　在出来的文本框里，依次输入 IP 地址、子网掩码、网关，然后按回车键确定，在下面的 DNS 里输入 DNS 服务器地址，检查一下，点右下角的“保存”按钮，在出来的认证中，输入自己的密码；

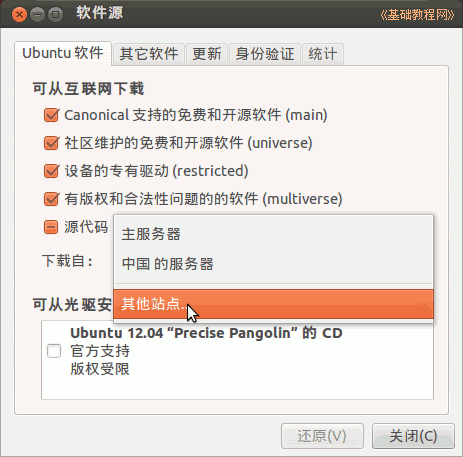


　回到原来对话框点“关闭”，稍等一会就可以上网了，也可以重新启动计算机；

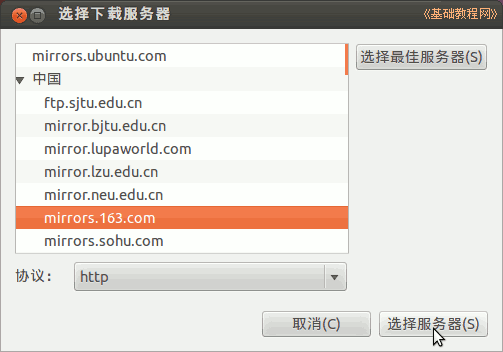
　8）连接好网络后先换源，在左边的快速启动面板中点击“软件中心”按钮，再把鼠标移到最上边出现菜单栏，点“编辑－软件源”；



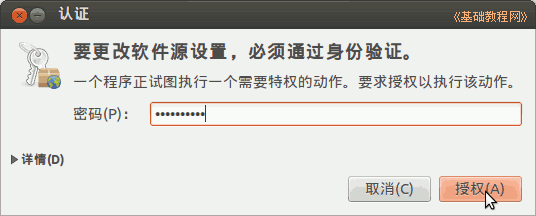
　9）在出来的软件源面板下边，点击“中国的服务器”下拉列表，选择“其他站点”；



　10）在出来的服务器列表中，选择搜狐、163 或 cn99 的站点都可以，然后点右下角的“选择服务器”按钮返回；



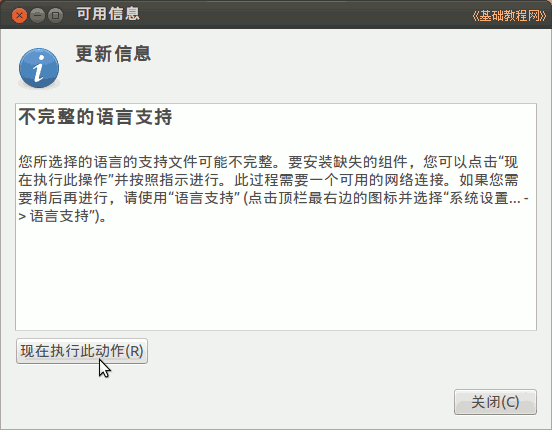
　11）关闭窗口后，出来身份验证，输入自己的登录密码后点“授权”，如果出来更新提示，点“重新载入”，等待完成就好了；



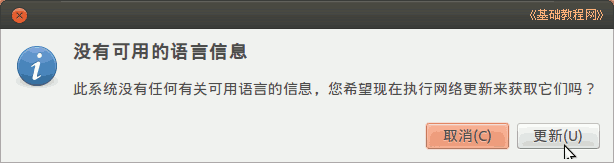
　更多设置源的相关介绍，请参阅：<http://teliute.org/linux/Ubsetup/lesson9/lesson9.html>

## 更新系统

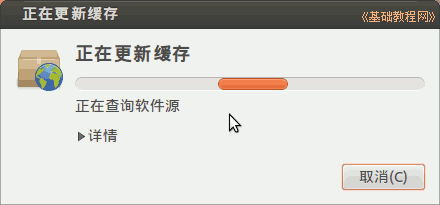
　1）回到最开头的“不完整语言言支持”对话框，点“现在执行此动作”，如果找不着了，就点右上角齿轮图标－系统设置－语言支持；



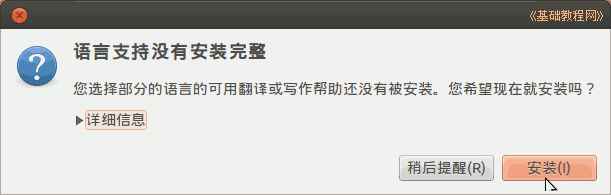
　2）如果出来提示“没有可用的语言信息”面板，点击“更新”按钮；



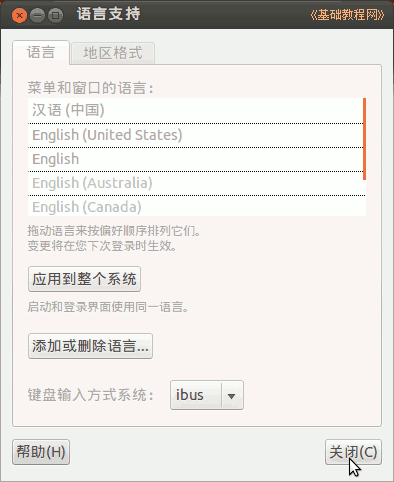
　3）然后出来一个缓存更新的对话框，耐心等待完成，点“详情”可以查看进度；



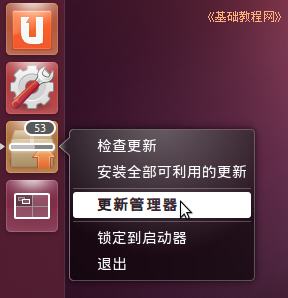
　4）然后在出来的“语言支持没有完全安装”对话框中，点“安装”，安装语言包；



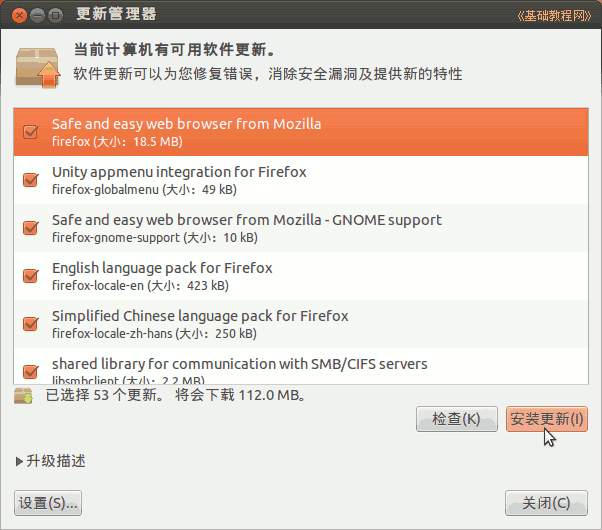
　5）待语言包安装完成后，点关闭按钮关闭对话框；



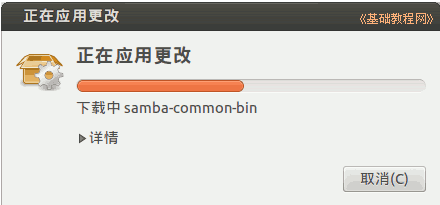
　6）在左边的快速启动面板中，还会有一个更新按钮(上边标有数字)，点击出来更新管理器窗口，或者点右键选“更新管理器”；

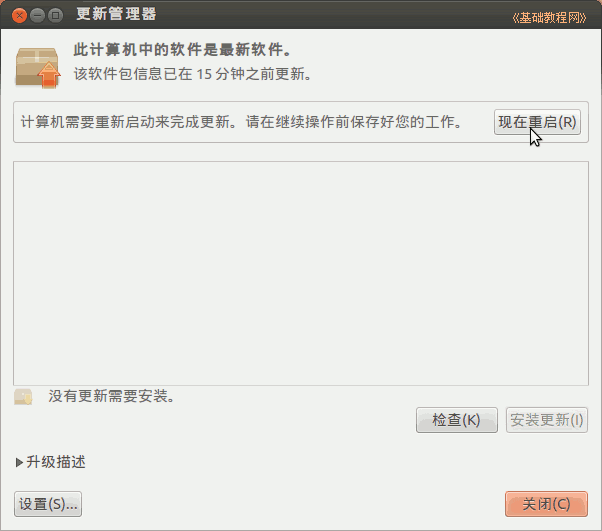
　7）在出来的窗口右下角，点击“安装更新”按钮，左边的“设置”里也可以设置软件源；



　8）接下来就是下载和安装，这儿耐心等待即可；



9）安装完成后，点右上角的“现在重启”按钮重启动电脑，然后就可以开始使用了；



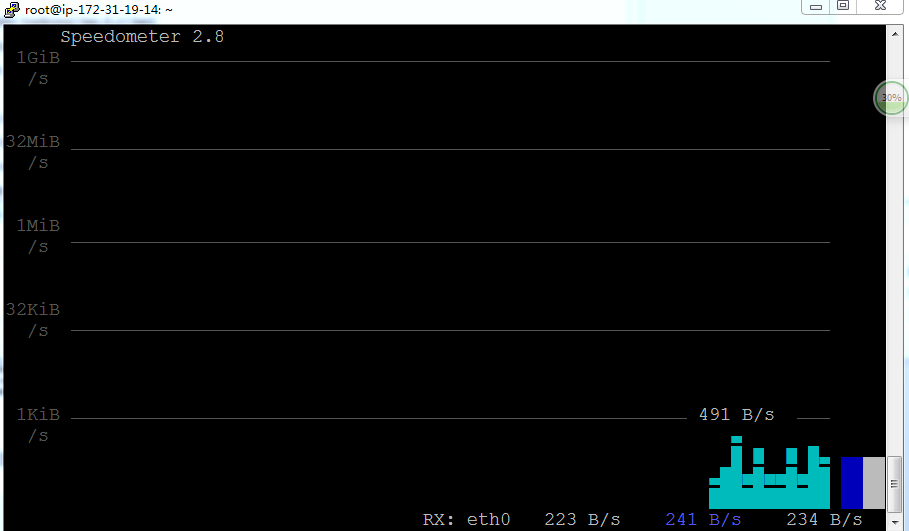
　其他细节：如果发现 Windows 的启动菜单项没了，先不着急，更新完就有了，如果还没有则在终端执行 sudo update-grub，或者把原来备份的 grub.cfg 找出来，照着填加进去；

# 附录[Ubuntu下的小工具](http://www.cnblogs.com/adforce/archive/2012/03/15/2399468.html)

## 测试工具speedometer

apt-get install speedometer

speedometer -rx eth0



## APT的使用

apt-get update

在修改/etc/apt/sources.list或/etc/apt/preferences之後运行该命令。此外您需要定期运行这一命令以确保您的软件包列表是最新的。

apt-get upgrade

可以使用这条命令更新软件包，apt-get upgrade不仅可以从相同版本号的发布版中更新软件包，也可以从新版本号的发布版中更新软件包，尽管实现后一种更新的推荐命令为apt-get dist-upgrade；

在运行apt-get upgrade命令时加上-u选项很有用（即：apt-get -u upgrade)。这个选项让APT显示完整的可更新软件包列表。不加这个选项，你就只能盲目地更新。APT会下载每个软件包的最新更新版本，然后以合理的次序安装它们。注意在运行该命令前应先运行 apt-get update更新数据库。更新任何已安装的软件包。[1]

apt-get dist-upgrade

将系统升级到新版本

apt-get install packagename

安装一个新软件包（参见下文的aptitude）

apt-get remove packagename

卸载一个已安装的软件包（保留配置文档）

apt-get --purge remove packagename卸载一个已安装的软件包（删除配置文档）

apt-get autoremove packagename

同上，比上面的要删除的彻底一点

dpkg --force-all --purge packagename

有些软件很难卸载，而且还阻止了别的软件的应用，就能够用这个，但是有点冒险。

apt-get autoclean

apt会把已装或已卸的软件都备份在硬盘上，所以假如需要空间的话，能够让这个命令来删除您已删掉的软件

apt-get clean

这个命令会把安装的软件的备份也删除，但是这样不会影响软件的使用。

apt-cache search string

在软件包列表中搜索字符串

dpkg -l package-name-pattern列出任何和模式相匹配的软件包。假如您不知道软件包的全名，您能够使用“\*package-name-pattern\*”。

aptitude

周详查看已安装或可用的软件包。和apt-get类似，aptitude能够通过命令行方式调用，但仅限于某些命令——最常见的有安装和卸载命令。

由于aptitude比apt-get了解更多信息，能够说他更适合用来进行安装和卸载。

apt-cache showpkg pkgs

显示软件包信息。

apt-cache dumpavail打印可用软件包列表。

apt-cache show pkgs

显示软件包记录，类似于dpkg –print-avail。

apt-cache pkgnames

打印软件包列表中任何软件包的名称。

dpkg -S file

这个文档属于哪个已安装软件包。

dpkg -L package

列出软件包中的任何文档。

dpkg -l

列出所有已安装的软件包

apt-file search filename

查找包含特定文档的软件包（不一定是已安装的），这些文档的文档名中含有指定的字符串。apt-file是个单独的软件包。您必须先使用apt-get install来安装他，然後运行apt-file update。假如apt-file search filename输出的内容太多，您能够尝试使用apt-file search filename | grep -w filename（只显示指定字符串作为完整的单词出现在其中的那些文档名）或类似方法，例如：apt-file search filename | grep /bin/（只显示位于诸如/bin或/usr/bin这些文档夹中的文档，假如您要查找的是某个特定的执行文档的话，这样做是有帮助的）

apt-get autoclean

定期运行这个命令来清除那些已卸载的软件包的.deb文档。通过这种方式，您能够释放大量的磁盘空间。假如您的需求十分迫切，能够使用apt-get clean以释放更多空间。这个命令会将已安装软件包裹的.deb文档一并删除。大多数情况下您不会再用到这些.debs文档，因此假如您为磁盘空间不足而感到焦头烂额，这个办法也许值得一试。

# 附录[Ubuntu下的SVN创建](http://www.cnblogs.com/adforce/archive/2012/03/15/2399468.html)

在[Ubuntu](http://www.linuxidc.com/topicnews.aspx?tid=2)下安装subversion，这里总结一下安装过程:首先要安装好**apache2**服务器,然后装**subversion**

·**apache2**是web服务器，用来通过web访问subversion

·**subversion**就是本文的主角

## ****安装subversion****

安装完apache2服务器后就开始安装subversion

sudo apt-get install subversion

sudo apt-get install subversion-tools

许多位置都可以放置 Subversion 文件仓库，其中两个最常用的是:/usr/local/svn 以及 /home/svn我创建在/hom/svn:

**创建目录**

//创建库文件所在的目录

mkdir /home/svn/repository

**创建用户**

$ sudo adduser svnuser

$ sudo addgroup subversion

$ sudo addgroup svnuser subversion

**创建项目目录**

$ sudo mkdir /home/svn

$ cd /home/svn

$ sudo chown -R root:subversion svn

$ sudo chmod -R g+rws svn

**创建Subversion库**

#创建仓库

svnadmin create /home/svn

cd /home/svn/

#看看是不是多了些文件，如果是则说明Subversion安装成功了

ls –l

**修改配置文件/var/svn/conf/svnserve.conf**

#去掉#[general]前面的#号

[general]

#匿名访问的权限，可以是read,write,none,默认为read

anon-access = none

#认证用户的权限，可以是read,write,none,默认为write

auth-access = write

#密码数据库的路径，去掉前面的#

password-db = passwd

#注意：所有的行都必须顶格，否则报错

**修改配置文件passwd**

[users]

svnuser = 123456

注意：

◆一定要去掉[users]前面的#,否则svn只能以匿名用户登录，客户端不会出现登录窗口，除非你的anon不为none,否则将返回一个错误。

◆这里的密码都是没有加密的

**停止Subversion服务器：**

killall svnserve

**启动Subversion服务器 对于单个代码仓库,启动命令：**

svnserve -d -r /var/svn --listen-host 10.10.10.10

其中-d表示在后台运行，-r指定服务器的根目录，这样访问服务器时就可以直接 用svn://服务器ip来访问了。

**扩展**

如果服务器有多ip的话–listen-host来指定监听 的ip地址. www.pgtohr.com我们可以在Subversion客户端中通过svn://10.10.10.10来访问Subversion服务器.

对于多个代码仓库,我们在启动时也可以用-r选项来指定服务器根目录,但访问时 需要写上每个仓库相对于svn根目录的相对路径.

比如,我们有两个代码仓库/var/vss/repo1 和/var/vss/repo2,我们用 svnserve -d -r /var/vss –listen-host 10.19.3.103来启动,那么在客户端访 问时可以用svn://10.19.3.103/repo1和svn: //10.19.3.103/repo2来分别访问两个项目。

## 配置文件例子

**编辑svnserve.conf文件**

password-db = password

authz-db = authz

//补充说明

# [general]

anon-access = read

auth-access = write

password-db = passwd

其中 anon-access 和 auth-access 分别为匿名和有权限用户的权限，默认给匿名用户只读的权限,但如果想拒绝匿名用户的访问，只需把 read 改成 none 就能达到目的。

**编辑/home/svnuser/etc/passwd :**

[users]

mirze = 123456

test1 = 123456

test2 = 123456

**编辑/home/svnuser/etc/authz**

[groups]

admin = mirze,test1

test = test2

[/]

@admin=rw

\*=r

这里设置了三个用户mirze,test1,test2密码都是123456

其中mirze和test1属于admin组，有读和写的权限,test2属于test组只有读的权限

## Hooks的使用

同步程序思路：用户提交程序到SVN，SVN触发hooks,按不同的hooks进行处理，这里用到的是post-commit，利用post-commit到代码检出到SVN服务器的本地硬盘目录，再通过rsync同步到远程的WEB服务器上。

1、SVN的hooks

post-commit

在提交完成，成功创建版本之后执行该钩子，提交已经完成，不可更改，因此本脚本的返回值被忽略。

post-lock

对文件进行加锁操作之后执行该脚本

post-revprop-change

在修改 revision 属性之后，执行该脚本。因为修改稿已经完成，不可更改，因此本脚本的返回值被忽略（不过实际上的实现似乎是该脚本的正确执行与否影响属性修改）

post-unlock

对文件进行解锁操作之后执行该脚本

pre-commit

在 Subversion transaction 完毕之后，在提交之前，执行该脚本

pre-lock

对文件进行加锁操作之前执行该脚本

pre-revprop-change

在修改 revision 属性之前，执行该脚本

pre-unlock

对文件进行解锁操作之前执行该脚本

start-commit

在客户端还没有向服务器提交数据之前，即还没有建立 Subversion transaction（缩写为 txn） 之前，执行执行该脚本

通过上面这些名称编写的脚本就就可以实现多种功能了，相当强大。

2、同步命令rsync的具体参数使用

3、具有基个语言的编程能力bash python perl都可以实现

1、post-commit脚本

sudo /usr/bin/svn update /tmp --username svnuser --password 123456

sudo /usr/bin/svn export "svn://localhost/" "/tmp/test" --force --username svnuser --password 123456

2、最简单的

SVN=/usr/local/subversion/bin/svn

WEB=/www/web

SVN update WEB

3、我们还可以再复杂一些,只有加入特殊的标注后再部署

#!/bin/sh

REPOS="$1"

REV="$2"

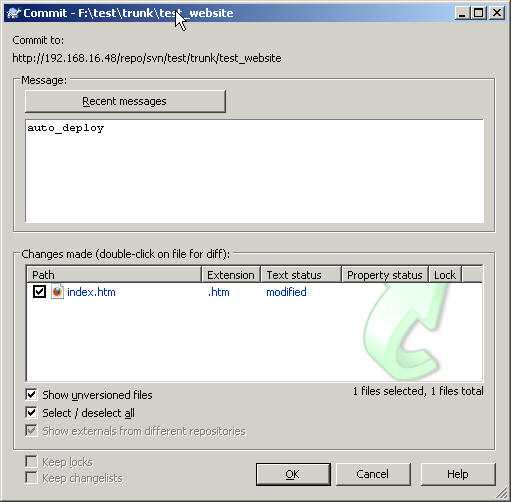
#当用户提交时注释中写上"auto\_deploy"的时候，就自动部署到apache目录，否则不部署。

if (svnlook log -r $REV /home/svn | grep "auto\_deploy" )

then

sudo /usr/bin/svn export "svn://localhost/" "/tmp/test" --force --username svnuser --password 123456

fi



# 附录Sublimetext的使用

## 安装Sublime Text 2插件的方法：

### 直接安装

安装Sublime text 2插件很方便，可以直接下载安装包解压缩到Packages目录

（菜单->preferences->packages）

### 使用Package Control组件安装

也可以安装package control组件，然后直接在线安装：

* 按Ctrl+`调出console
* 粘贴以下代码到底部命令行并回车：

import urllib2,os;pf='Package Control.sublime-package';ipp=sublime.installed\_packages\_path();os.makedirs(ipp) if not os.path.exists(ipp) else None;open(os.path.join(ipp,pf),'wb').write(urllib2.urlopen('http://sublime.wbond.net/'+pf.replace(' ','%20')).read())

* 重启Sublime Text 2。
* 如果在Perferences->package settings中看到package control这一项，则安装成功。

如果这种方法不能安装成功，可以

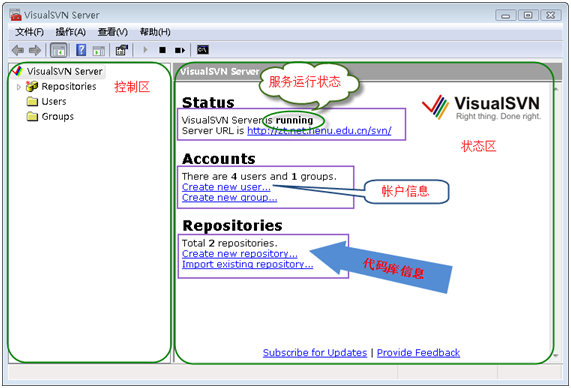
http://wbond.net/sublime\_packages/package\_control/installation

# 附录Windows下使用SVN Server

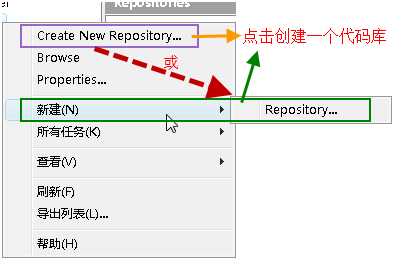
转载之jiayongchao的博客

## VisualSVN Server的配置和使用方法【服务器端】

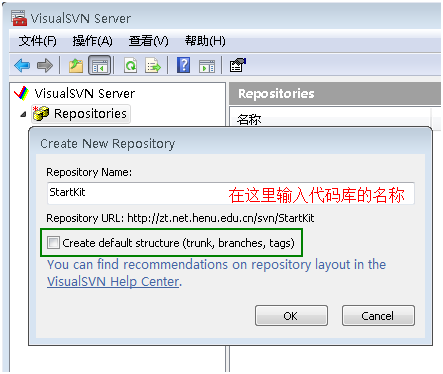
安装好VisualSVN Server后【安装过程看[这里](http://www.cnblogs.com/ttzhang/archive/2008/11/03/1325102.html)】，运行VisualSVN Server Manger，下面是启动界面：



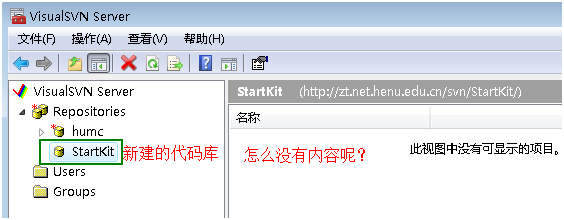
好的，下面我来添加一个代码库【Repository】，如下图：



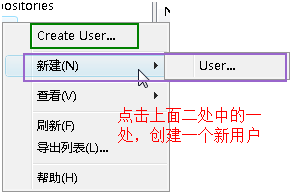
按上图所示，创建新的代码库，在下图所示的文本框中输入代码库名称：



注意：上图中的CheckBox如果选中，则在代码库StartKit下面会创建trunk、branches、tags三个子目录；不选中，则只创建空的代码库StartKit。点击OK按钮，代码库就创建成功了。

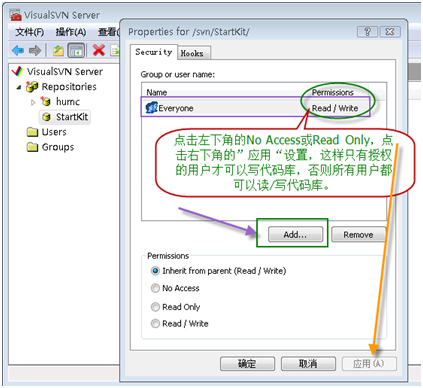


建完代码库后，没有任何内容在里面。我会在这个教程的第二部分说明如何迁入源代码。下面，我们开始安全性设置，在左侧的Users上点击右键：

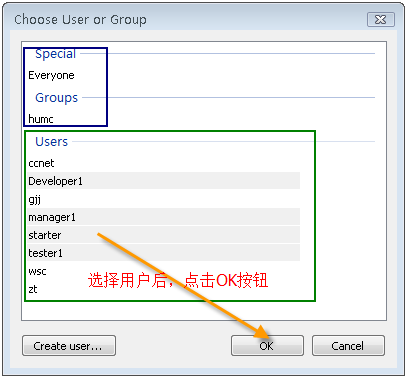




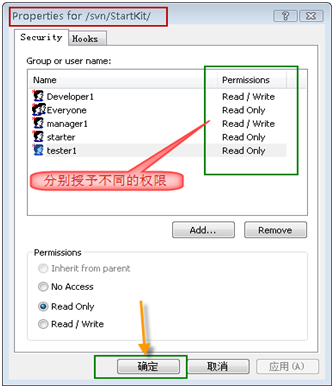
输入上面的信息，点击OK，我们就创建一个用户了。按照上面的过程，分别添加用户Developer1、tester1、manager1，好了，我们开始添加这些用户到我们刚才创建的项目里：



点击上图中的"Add..."按钮，在下图中选择我们刚才添加的用户,点击OK按钮：说明：大家可能注意到了下图中的Groups，是的，你也可以先创建组，把用户添加到各个组中，然后对组进行授权，操作比较简单，在此略过。



按照下图所示，分别对用户【或组】进行授权：



点击"确定"按钮，上面的用户就具有了访问StartKit代码库的不同权限。因为用户starter在团队中是新来者，不希望他向代码库中提交新代码，所以他只能读取代 码库中的代码，不能提交代码。tester1是测试人员，不负责代码编写，所以也是只读权限。而Developer1和manager1是开发人员和项目 经理，自然具有读、写的权限。

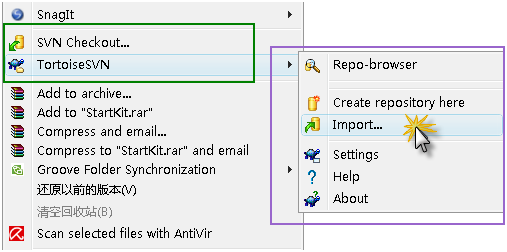
在实际的项目开发过程中，Developer和tester往往不可能只有一个人，这时候使用组来授权更加方便，这个大家可以自己练习一下。

## TotoiseSVN的基本使用方法

一、签入源代码到SVN服务器

假如我们使用Myeclipes在文件夹StartKit中创建了一个项目，我们要把这个项目的源代码签入到SVN Server上的代码库中里，首先右键点击StartKit文件夹，这时候的右键菜单如下图所示：

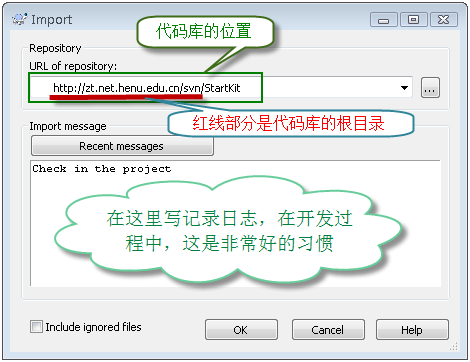
图2-2-1



点击Import，弹出下面的窗体，其中[http://zt.net.henu.edu.cn](http://zt.net.henu.edu.cn/) 是服务器名，svn是代码仓库的根目录，StartKit是我们在上个教程中添加的一个代码库：

说明：左下角的CheckBox，在第一次签入源代码时没有用，但是，在以后你提交代码的时候是非常有用的。

图2-2-2



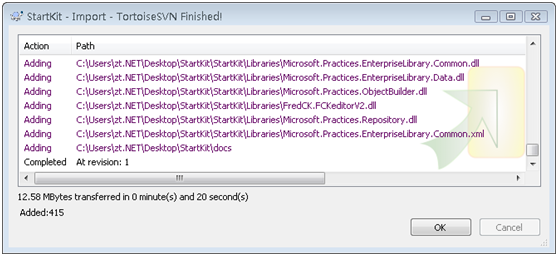
点击OK按钮，会弹出下面的窗体，要求输入凭据：

图2-2-3



在上面的窗体中输入用户名和密码，点击OK按钮：

图2-2-4



如上图所示，好了，源代码已经成功签入SVN服务器了。这时候团队成员就可以迁出SVN服务器上的源代码到自己的机器了。

# 附录[vmware中的bridge、nat区别](http://www.cnblogs.com/adforce/archive/2012/03/15/2399468.html)

概述：

VMWare提供了三种工作模式，它们是bridged(桥接模式)、NAT(网络地址转换模式)和host-only(主机模式)。要想在网络管理和维护中合理应用它们，你就应该先了解一下这三种工作模式。

## ****bridged(桥接模式)****

　　在这种模式下，VMWare虚拟出来的操作系统就像是局域网中的一台独立的主机，它可以访问网内任何一台机器。

　　在桥接模式下，你需要手工为虚拟系统配置IP地址、子网掩码，而且还要和宿主机器处于同一网段，这样虚拟系统才能和宿主机器进行通信。同时，由于这个虚拟系统是局域网中的一个独立的主机系统，那么就可以手工配置它的TCP/IP配置信息，以实现通过局域网的网关或路由器访问互联网。

　　使用桥接模式的虚拟系统和宿主机器的关系，就像连接在同一个Hub上的两台电脑。想让它们相互通讯，你就需要为虚拟系统配置IP地址和子网掩码，否则就无法通信。

　　如果你想利用VMWare在局域网内新建一个虚拟服务器，为局域网用户提供网络服务，就应该选择桥接模式。

## ****host-only(主机模式)****

　　在某些特殊的网络调试环境中，要求将真实环境和虚拟环境隔离开，这时你就可采用host-only模式。在host-only模式中，所有的虚拟系统是可以相互通信的，但虚拟系统和真实的网络是被隔离开的。

　　提示：在host-only模式下，虚拟系统和宿主机器系统是可以相互通信的，相当于这两台机器通过双绞线互连。

　　在host-only模式下，虚拟系统的TCP/IP配置信息(如IP地址、网关地址、DNS服务器等)，都是由VMnet1(host-only)虚拟网络的DHCP服务器来动态分配的。

　　如果你想利用VMWare创建一个与网内其他机器相隔离的虚拟系统，进行某些特殊的网络调试工作，可以选择host-only模式。

## ****NAT(网络地址转换模式)****

　　使用NAT模式，就是让虚拟系统借助NAT(网络地址转换)功能，通过宿主机器所在的网络来访问公网。也就是说，使用NAT模式可以实现在虚拟系统里访问互联网。NAT模式下的虚拟系统的TCP/IP配置信息是由VMnet8(NAT)虚拟网络的DHCP服务器提供的，无法进行手工修改，因此虚拟系统也就无法和本局域网中的其他真实主机进行通讯。采用NAT模式最大的优势是虚拟系统接入互联网非常简单，你不需要进行任何其他的配置，只需要宿主机器能访问互联网即可。

　　如果你想利用VMWare安装一个新的虚拟系统，在虚拟系统中不用进行任何手工配置就能直接访问互联网，建议你采用NAT模式。

　　提示：以上所提到的NAT模式下的VMnet8虚拟网络，host-only模式下的VMnet1虚拟网络，以及bridged模式下的VMnet0虚拟网络，都是由VMWare虚拟机自动配置而生成的，不需要用户自行设置。VMnet8和VMnet1提供DHCP服务，VMnet0虚拟网络则不提供。

这个问题一直都没有彻底的搞清楚，心里不是很有数，Vmware中什么网桥，NAT啊，Hostonly什么的，另外还有Colinux里面的网络问题。总之很繁杂，搞清楚很有必要。

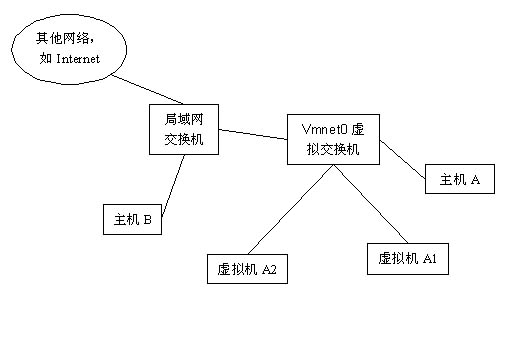
## ****VMware网络连接的几种方式****

大家在安装完虚拟机后，默认安装了两个虚拟网卡，VMnet1和 VMnet8，其他的未安装（当然也可以手动安装其他的）。其中VMnet1是host网卡，用于host方式连接网络的。VMnet8是NAT网卡，用 于NAT方式连接网络的。它们的IP地址是随机生成的，如果要用虚拟机做实验的话，最好将VMnet1到VMnet8的IP地址改掉。习惯上把 VMware虚拟网卡使用的网段“固定”，使用如下原则：VMnet1对应的网段是192.168.10.0，VMnet2对应的网段是 192.168.20.0，其他的类似。当然平常只是随便用用的就不用改了，能上网就行了。

网络连接的方式主要有：Bridged（桥接）、NAT、host（主机网络）。

**Use bridged networking（使用桥接网络）**

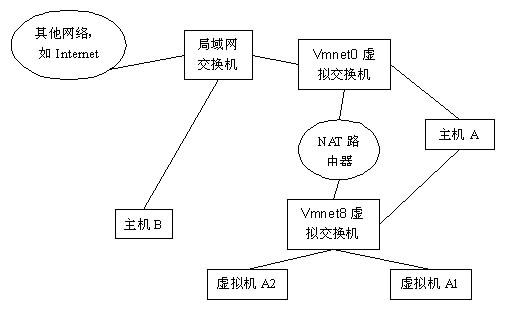
说明：使用VMnet0虚拟交换机，此时虚拟机相当与网络上的一台独立计算机与主机一样，拥有一个独立的IP地址。



使用桥接方式，A，A1，A2，B可互访。

**Use network address translation（NAT）**

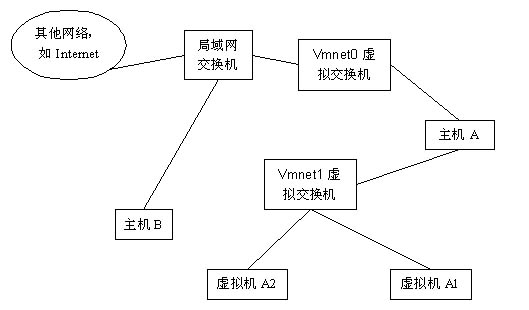
说明：使用Vmnet8虚拟交换机，此时虚拟机可以通过主机单向网络上的其他工作站，其他工作站不能访问虚拟机。



使用NAT方式，A1，A2可以访问B，但B不可以访问A1，A2。但A，A1，A2可以互访。

**Use Host-Only networking（使用主机网络）**

说明：使用Vmnet1虚拟交换机，此时虚拟机只能与虚拟机、主机互访。也就是不能上Internet



使用Host方式，A，A1，A2可以互访，但A1，A2不能访问B，也不能被B访问。

    实际应用：现在最普遍使用的是宽带，使用桥接方式，主机必然可以上网，但是虚拟机上的操作系统是不可以上网的。此时你的主机就相当于A，而虚拟机就相当于 A1，除非你再向运营商（如电信，网通）申请个IP地址，显然这是不可取 的。如果你有路由器的话，那么就可以使用桥接方式上网，因为此时你申请的IP地址是写在了路由器上，而不是你的机器上，这样包括你的主机，虚拟机，也包括 连在路由器上的其他人的机器，都将可以上网，使用着由路由器自动分配的IP地址，IP地址一般是192.168.0.XXX。貌似运营商不太乐意你使用路由器，说什么对他们交换机的接口有损害。当然，只要你的主机可以上网，使用NAT方式，虚拟机总是可以上网（当然不包括在你胡乱改写IP，网关，DNS等 信息的情况下）。

    实例手动设置：假设主机的VMnet1使用网段地址为192.168.10.0，VMnet8使用网段地址为192.168.80.0，网关地址为 192.168.80.254（也就是NAT路由器地址，可自己设置，在VMnet8上），主机网卡使用地址为192.168.1.1。

一、使用桥接方式，虚拟机A1的IP地址可以设置成192.168.1.5（与主机网卡地址同网段的即可），其他的诸如网关地址，DNS，子网掩码均与主机的相同。

二、使用NAT方式，虚拟机A1的IP地址可以设置成192.168.80.5（与VMnet8使用相同的网段），网关是NAT路由器地址，即192.168.80.524

三、使用host-only方式，虚拟机A1的IP地址可以设置成192.168.10.5（与VMnet1使用相同的网段）

说明：一般用户不需要自己手动设置，选择自动获取，或开启DHCP服务即可。

**2.第二套说法**

很多朋友都用vmware来测试不同的系统，我结合自己的经验谈一下对网络设置的理解，不对的地方请指正。

bridge：

这种方式最简单，直接将虚拟网卡桥接到一个物理网卡上面，和linux下一个网卡 绑定两个不同地址类似，实际上是将网卡设置为混杂模式，从而达到侦听多个IP的能力。

在此种模式下，虚拟机内部的网卡（例如linux下的eth0)直接连到了物理网卡所在的网络上，可以想象为虚拟机和host机处于对等的地位，在网络关系上是平等的，没有谁在谁后面的问题。

使用这种方式很简单，前提是你可以得到1个以上的地址。对于想进行种种网络实验的朋友 不太适合，因为你无法对虚拟机的网络进行控制，它直接出去了。

nat方式：

这种方式下host内部出现了一个虚拟的网卡vmnet8（默认情况下），如果你有过 做nat服务器的经验，这里的vmnet8就相当于连接到内网的网卡，而虚拟机本身则相当于运 行在内网上的机器，虚拟机内的网卡（eth0）则独立于vmnet8。

你会发现在这种方式下，vmware自带的dhcp会默认地加载到vmnet8界面上，这样 虚拟机就可以使用dhcp服务。更为重要的是，vmware自带了nat服务，提供了从vmnet8到外网的地址转 换，所以这种情况是一个实实在在的nat服务器在运行，只不过是供虚拟机用的。

很显然，如果你只有一个外网地址，此种方式很合适。

hostonly:

这应该是最为灵活的方式，有兴趣的话可以进行各种网络实验。和nat唯一的不同的是，此 种方式下，没有地址转换服务，因此，模认情况下，虚拟机只能到主机访问，这也是hostonly的名字的意义。

默认情况下，也会有一个dhcp服务加载到vmnet1上。这样连接到vmnet1上的虚拟机仍然可以设置成dhcp，方便系统的配置.

是不是这种方式就没有办法连接到外网呢，当然不是，事实上，这种方式更为灵活，你可以使用自己的方式，从而达到最理想的配置，例如：

a。使用自己dhcp服务：首先停掉vmware自带的dhcp服务，使dhcp服务更为统一。

b。使用自己的nat,方便加入防火墙。windows host可以做nat的方法很多，简单的如windows xp的internet共享，复杂的如windows server里的nat服务。

c. 使用自己的防火墙。因为你可以完全控制vmnet1,你可以加入（或试验）防火墙在vmnet1和外网的网卡间。

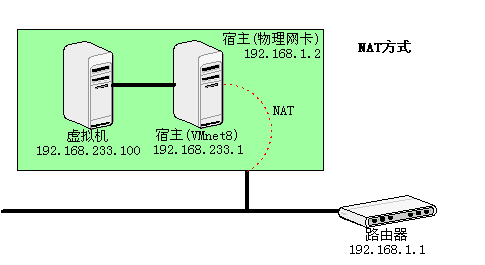
从以上可以看出，hostonly这种模式和普通的nat server带整个内网上网的情形类似，因此你可以方便的进行与之有关的实验，比如防火强的设置等。

**3.第三套说法**

VMware支持三种类型的网络：NAT，Bridged，Host-only。

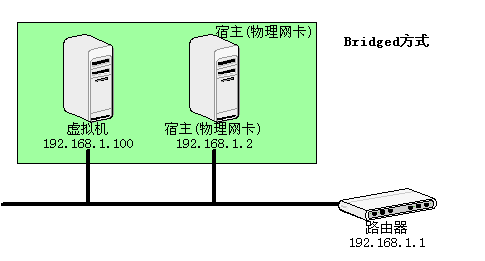
NAT

这种方式下，虚拟机的网卡连接到宿主的 VMnet8 上。此时系统的 VMWare NAT Service 服务就充当了路由器的作用，负责将虚拟机发到 VMnet8 的包进行地址转换之后发到实际的网络上，再将实际网络上返回的包进行地址转换后通过 VMnet8 发送给虚拟机。VMWare DHCP Service 负责为虚拟机提供 DHCP 服务。



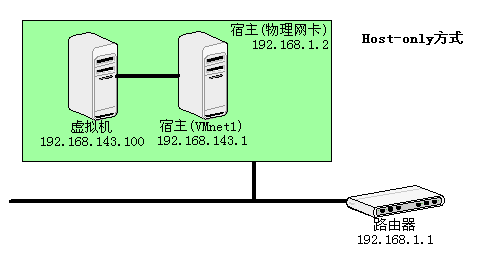
Bridged

这种方式下，虚拟机就像一台真正的计算机一样，直接连接到实际的网络上，与宿主机没有任何联系。



Host-only

这种方式下，虚拟机的网卡连接到宿主的 VMnet1 上，但系统并不为虚拟机提供任何路由服务，因此虚拟机只能和宿主机进行通信，而不能连接到实际网络上。



# 附录 Python学习资源大全

强烈推荐-[**易枭寒**](http://weibo.com/1866518221)大侠的学习资源整理

本文 **主要** 针对 **零基础** 想学习Python者。本文本文由@史江歌（[shijiangge@gmail.com](mailto:shijiangge@gmail.com)）根据互联网资料收集整理而成，感谢互联网，感谢各位的分享。鸣谢！本文会不断更新。

## 学会使用搜索引擎，尤其是Google。推荐阅读文章：

[《如何用好Google搜索引擎？》](http://www.zhihu.com/question/20161362) --知乎问答@崔凯

[《十大高明的Google搜索技巧》](http://www.williamlong.info/archives/728.html)by-雪影蓝风

《如何用好Google》by-李笑来

## 学会提问

推荐阅读《提问的智慧》--by [啄木鸟社区](http://wiki.woodpecker.org.cn/moin/AskForHelp)

## 书籍推荐：

1. Python官方文档 [点击打开链接](http://docs.python.org/2/tutorial/index.html) 英语不好的，暂时就先别看了，以后一定肯定要加强英语学习哦;)。
2. 对于零基础者，推荐《How to think like a computer scientist》[点击打开链接](http://www.greenteapress.com/thinkpython/thinkCSpy.pdf)，对应中文版本《Python学习笔记--皮大庆》，非常适合零基础入门。
3. 对于零基础者，推荐《Python编程实践》 [点击打开链接](http://book.douban.com/subject/7059900/)
4. 对于零基础者，推荐《简明Python教程》 [点击打开链接](http://woodpecker.org.cn/abyteofpython_cn/chinese/)
5. 对于零基础者，推荐《笨办法学 Python （Learn Python The Hard Way）》[点击打开链接](http://learn-python-the-hard-way-zh_cn-translation.readthedocs.org/en/1.0/)
6. 对于有其他编程语言基础者，推荐《Python基础教程》[点击打开链接](http://book.douban.com/subject/4866934/)
7. 更多书籍推荐， **知乎问答**：关于 Python 的经典入门书籍有哪些？[点击打开链接](http://www.zhihu.com/question/19593179)
8. 更多书籍推荐， **V2EX问答**：文科生想学Python [点击打开链接](http://www.v2ex.com/t/40397)

## 视频推荐：

麻省理工公开课《计算机科学及编程导论》，该课程针对零基础者 [点击打开链接](http://v.163.com/special/opencourse/bianchengdaolun.html) 网易公开课。

课程介绍 这门课程适用于那些拥有 **很少或没有编程经验** 的学生,它致力于使学生理解计算机在解决问题中的作用,并且帮助学生， **不论其专业**，使他们对于能够完成有用的小程序的目标充满信心。

## 学习方法

### 知乎问答：零基础，应当如何开始学习 Python ？[点击打开链接](http://www.zhihu.com/question/20039623?nr=1)--by@黎敏

虽然我不是Python高手，但我是零基础，之前会的都是软件PS，PPT之类。

如果目的是想成为程序员，参考教学大纲。

如果只是学程序，理解科技，解决工作问题，我的方式可以参考使用：

1. 找到合适的入门书籍，大致读一次，循环啊判断啊，常用类啊，搞懂（太难的跳过）
2. 做些简单习题，字符串比较，读取日期之类 《Python Cookbook》不错（太难太无趣的，再次跳过，保持兴趣是最重要的，不会的以后可以再学）
3. 加入Python讨论群，态度友好笑眯眯（很重要，这样高手才会耐心纠正你错误常识）。 很多小问题，纠结许久，对方一句话点播思路，真的节约你很多时间。耐心指教我的好人，超级超级多谢。
4. 解决自己电脑问题。 比如下载美剧，零散下载了2，4，5，8集，而美剧共12集，怎样找出漏下的那几集？然后问题分解，1读取全部下载文件名，2提取集的数字，3数字排序和（1--12）对比，找出漏下的。
5. 时刻记住目的，不是为了当程序员，是为了解决问题。

比如，想偷懒抓网页内容，用urllib不行，用request也不行，才发现抓取内容涉及那么多方面（cookie，header，SSL，url，javascript等等），当然可以听人家劝，回去好好读书，从头读。

或者，不求效率，只求解决，用ie打开网页再另存为行不行？ie已经渲染过全部结果了。

问题变成：1--打开指定的10个网页（一行代码就行）。更复杂的想保存呢？利用已经存在的包，比如PAM30（我的是Python3），直接打开ie，用函数outHTML另存为文本，再用搜索函数（str搜索也行，re正则也行） 找到数据。简单吧？而且代码超级短。

1. 保持兴趣，用最简单的方式解决问题，什么底层驱动，各种交换，留给大牛去写吧。我们利用已经有的包完成。
2. 耐心读文档，并且练习快速读文档。拿到新包，找到自己所需要的函数，是需要快速读一次的。这个不难，读函数名，大概能猜到是干嘛的，然后看看返回值，能判断是不是自己需要的。
3. 写帮助文件和学习笔记，并发布共享。教别人的时候，其实你已经自己再次思考一次了。

我觉得学程序就像学英文，把高频率的词（循环，判断，常用包，常用函数）搞懂，就能拼装成自己想要的软件。

然后，[http://stackoverflow.com](http://stackoverflow.com/) 和 [http://Google.com](http://google.com/) 是很好用的。

然后，坚持下去。

一定要保持兴趣，太复杂的跳过，就像小学数学，小学英语，都是由简入深。

网络很平面，无数国际大牛著作好书，关于Python，算法，电脑，网络，或者程序员思路，或者商业思维（浪潮之巅是本好书）等等，还有国际名校的网络公开课（中英文字幕翻译完毕，观看不是难事），讲计算机，网络，安全，或者安卓系统，什么都有，只要能持续保持兴趣，一点点学习下去，不是难事。

所有天才程序员，都曾是儿童，回到儿童思维来理解和学习。觉得什么有趣，先学，不懂的，先放着，遇到问题再来学，效果更好。

唯一建议是，不要太贪心，耐心学好一门优雅的语言，再学其它。虽然Javascript做特效很炫，或提某问题时，有大牛建议，用Ruby来写更好之类，不要改方向。就像老笑话：“要学习递归，必须首先理解递归。”然后死循环一直下去。坚持学好一门语言，再研究其他。

即使一门语言，跟网络，数据库等等相关的部分，若都能学好，再学其他语言，是很快的事情。

另外就是，用学英文的耐心来学计算机，英文遇到不懂的词，抄下，查询。

python里，看到Http，查查定义，看到outHtml，查查定义，跟初学英语时候一样，不要直接猜意思，因为精确描述性定义，跟含糊自然语有区别的。而新人瞎猜，很容易错误理解，wiki，google很有用。

### 芝麻问答：如何学习Python [点击打开链接](http://www.zhimaq.com/questions/70/python)作者：@halida

#### 如何学习python

我们假设你是一个初级程序员, 只懂得一点点的基础知识, 希望能够用python来做开发. 这篇文档就是为了满足以上目标而写的.

**大纲**

按照这篇文档所指示的任务过一遍, 你就能够做到:

熟悉python语言, 以及学会python的编码方式. 熟悉python库, 遇到开发任务的时候知道如何去找对应的模块. 知道如何查找和获取第三方的python库, 以应付开发任务. 学习步骤

**安装开发环境**

如果你在window下, 去下载pythonxy安装起来, 然后运行python, 进入python解释环境.

如果你在ubuntu下, 执行: sudo apt-get install python, 然后在命令行下运行python, 进入python解释环境.

**学习方法**

作为一名成熟的开发人员, 我学习新东西(假设是pyqt)的习惯方式是:

直接用google搜索pyqt的官方网站. 按照官方网站的说明, 下载pyqt. (如果是用ubuntu, 看看软件库里面是否有足够新的版本) 下载过程中, 开始阅读官方网站上面的教程. 一边看教程, 一遍按照教程使用pyqt. 如果发现教程不够全面, 用google搜索是否有对应的教学书籍可以看. 示例学习完毕, 开发一个玩具程序, 用来检验自己是否需要用到的功能. 开始学习python

我建议你学习的过程也按照上面来, 首先过一遍python官方文档:

<http://docs.python.org/tutorial/index.html>

然后做 <http://www.pythonchallenge.com/> 这个网站上面的题目.

如果卡在某一关太久, 可以看答案(google python challenge answer), 做完后看看别人的编码方式和自己有什么区别.

**小项目**

做完一遍后, 你会发现已经熟悉了基本的python开发. 然后做点小项目吧. 这里是一些题目, 挑感兴趣的去做.

* 写一个简单的计算器/记账软件/扫雷游戏(用pyqt库做界面)
* 写一个聊天室网站(用webpy框架, jquery刷新新的回复)
* 写一个爬虫, 获取douban上面所有用户的地点, 画地点分布的直方图(用lxml解析, 保存数据到sqlite里面去, 用matplotlib画图)

**FAQ**

Q: 遇到了问题, 到哪里求助?

A: 上 <http://groups.google.com/group/python-cn> (需要翻墙) 或者 [http://stackoverflow.com](http://stackoverflow.com/) 提问就可以了.

Q: 如何查找python的某个功能?

A: 看官方文档. <http://docs.python.org/library/index.html>

Q: 如何用python完成一个任务(比如写网站)?

A: google: python 写网站, 或者 google: python web development.

### 知乎问答：你是如何自学 Python 的？[点击打开链接](http://www.zhihu.com/question/20702054) 作者:@成增存

**阶段一**：前辈领进门。 第一眼是一个前辈给我看了看Python的代码，因为自己最早学习的是Java语言，第一眼看到Python的时候，前辈说，“Python是面向对象的”，然后就 打印了几句代码。可我怎么也看不出到底哪里是“面向对象”的。前辈说“在Python里，一切皆对象”，我才有点领悟的说道：“原来把什么看成对象，就是面向对象”，哈哈。

**阶段二**：开始看的是《Dive into Python》、《Leaning python》、《Python Doc》。 因为有很多语言学习经历，很快就完成了初步语法和使用学习。太复杂的特性还是在使用中逐步掌握的。

**阶段三**：开始使用Python做自己的一些日常工作。 比如Python搜索文件，Python批量处理等，使用最多的还是re模块和socket相关模块。写了大量的例子，让自己对Python更加喜爱，也更加熟悉。此时翻阅最多的是《Python Doc》的指南。不管是语言参考、库参考、Demo参考，都有大量可使用的内容，内容质量很高堪比JDK。

**阶段四**：生产上马。 开始使用Django，Flask，Tornado开发一些web应用，写一些日常使用的工具包等。逐步提升设计能力，和整体代码的管理能力。

**阶段五**：更合理的分配好C，Java，Python三门语言各自擅长的部分。 把合适的语言用到合适的地方。尽管一门语言有时候可以搞定所有的，但用擅长的语言解决合适的问题才是效率最高的。这也是“Python的大道至简”的理念带给我的帮助和认识。

PS：其中过程中主要的一些 **方法** :

1. 看书。学习的基础。
2. 自己本地练习。编程还是要实践出真知。
3. 资料查询。google，stackoverflow等多关注。
4. 交流。各种论坛上的python group，论坛。最早去的CU，JavaEye，不过现在貌似去的少了。\*\*google group\*\* 必须订阅。

学习+实践+总结，掌握语言的法宝。

## Just Do It

看了这么多资料了，还没动手敲代码？

好吧，最好一篇文章《我在学习编程中犯的两个 **最大错误** 》[点击打开链接](http://blog.jobbole.com/26552/)

Just do it 。动手敲代码吧。

注：本文由@史江歌（[shijiangge@gmail.com](mailto:shijiangge@gmail.com)）根据互联网资料收集整理而成。

## 更新

Python技术博客、招聘、开源软件、Python中文网站导航 [点击打开链接](http://simple-is-better.com/sites/)

遇到python问题怎么样解决？python help dir stackoverflow docs google [点击打开链接](http://blog.csdn.net/xiaowanggedege/article/details/8753013)

在线视频教学(英文) [点击打开codecademy](http://www.codecademy.com/zh/tracks/python)

英文[pythonforbeginners](http://www.pythonforbeginners.com/)

英文How do I learn Python from zero to web development?（各种电子书推荐）[点击打开链接](http://programmers.stackexchange.com/questions/12189/how-do-i-learn-python-from-zero-to-web-development)

# 附录-常用Python模块

Python具有强大的扩展能力，我列出了50个很棒的Python模块，包含几乎所有的需要：比如Databases，GUIs，Images, Sound, OS interaction, Web,以及其他。推荐收藏。

Graphical interface wxPython http://wxpython.org

Graphical interface pyGtk http://www.pygtk.org

Graphical interface pyQT http://www.riverbankcomputing.co.uk/pyqt/

Graphical interface Pmw http://pmw.sourceforge.net/

Graphical interface Tkinter 3000 http://effbot.org/zone/wck.htm

Graphical interface Tix http://tix.sourceforge.net/

Database MySQLdb http://sourceforge.net/projects/mysql-python

Database PyGreSQL http://www.pygresql.org/

Database Gadfly http://gadfly.sourceforge.net/

Database SQLAlchemy http://www.sqlalchemy.org/

Database psycopg http://www.initd.org/pub/software/psycopg/

Database kinterbasdb http://kinterbasdb.sourceforge.net/

Database cx\_Oracle http://www.cxtools.net/default.aspx?nav=downloads

Database pySQLite http://initd.org/tracker/pysqlite

MSN Messenger msnlib http://auriga.wearlab.de/~alb/msnlib/

MSN Messenger pymsn http://telepathy.freedesktop.org/wiki/Pymsn

MSN Messenger msnp http://msnp.sourceforge.net/

Network Twisted http://twistedmatrix.com/

Images PIL http://www.pythonware.com/products/pil/

Images gdmodule http://newcenturycomputers.net/projects/gdmodule.html

Images VideoCapture http://videocapture.sourceforge.net/

Sciences and Maths scipy http://www.scipy.org/

Sciences and Maths NumPy http://numpy.scipy.org/

Sciences and Maths numarray http://www.stsci.edu/resources/software\_hardware/numarray

Sciences and Maths matplotlib http://matplotlib.sourceforge.net/

Games Pygame http://www.pygame.org/news.html

Games Pyglet http://www.pyglet.org/

Games PySoy http://www.pysoy.org/

Games pyOpenGL http://pyopengl.sourceforge.net/

Jabber jabberpy http://jabberpy.sourceforge.net/

Web scrape http://zesty.ca/python/scrape.html

Web Beautiful Soup http://crummy.com/software/BeautifulSoup

Web pythonweb http://www.pythonweb.org/

Web mechanize http://wwwsearch.sourceforge.net/mechanize/

Localisation geoname.py http://www.zindep.com/blog-zindep/Geoname-python/

Serial port pySerial http://pyserial.sourceforge.net/

Serial port USPP http://ibarona.googlepages.com/uspp

Parallel Port pyParallel http://pyserial.sourceforge.net/pyparallel.html

USB Port pyUSB http://bleyer.org/pyusb/

Windows ctypes http://starship.python.net/crew/theller/ctypes/

Windows pywin32 http://sourceforge.net/projects/pywin32/

Windows pywinauto http://www.openqa.org/pywinauto/

Windows pyrtf http://pyrtf.sourceforge.net/

Windows wmi http://timgolden.me.uk/python/wmi.html

PDA/GSM/Mobiles pymo http://www.awaretek.com/pymo.html

PDA/GSM/Mobiles pyS60 http://sourceforge.net/projects/pys60

Sound pySoundic http://pysonic.sourceforge.net/

Sound pyMedia http://pymedia.org/

Sound FMOD http://www.fmod.org/

Sound pyMIDI http://www.cs.unc.edu/Research/assist/developer.shtml

GMail libgmail http://libgmail.sourceforge.net/

Google pyGoogle http://pygoogle.sourceforge.net/

Expect pExpect http://pexpect.sourceforge.net/

WordNet pyWordNet http://osteele.com/projects/pywordnet/

Command line cmd http://blog.doughellmann.com/2008/05/pymotw-cmd.html

Compiler backend llvm-py http://mdevan.nfshost.com/llvm-py/

3D VPython http://vpython.org